LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

PROGRAMACIÓN EN JAVASCRIPT

GESTIÓN DE EVENTOS • SCRIPTS INCRUSTADOS
COMPARATIVA JAVA, CGI

AÑO 3. Nº 30
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$

MULTITAREA EN TAPI

DEMOSCENE: MODELOS DE ILUMINACIÓN

BORLAND C++ 5.01

Y ADEMÁS

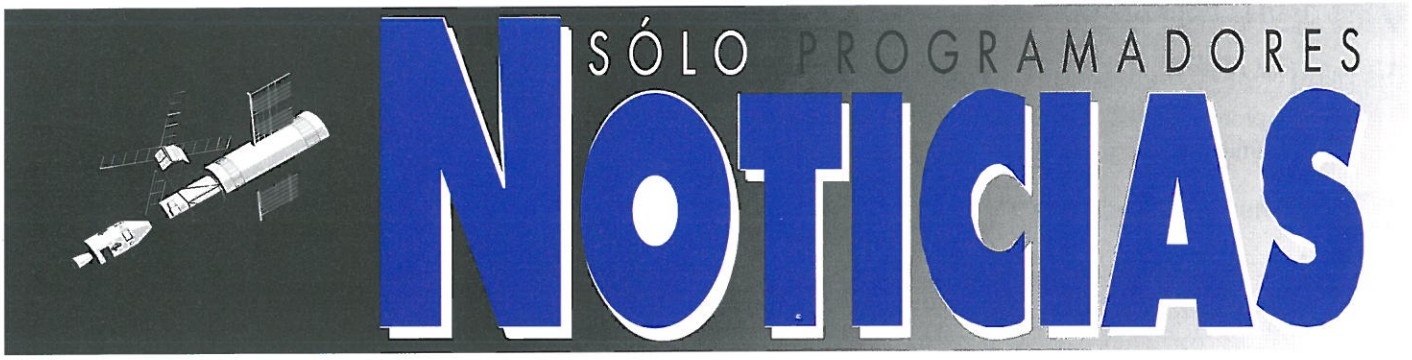
- Visual C++
- Protocolos IPX/SPX
- Grandes sistemas
- Inteligencia artificial

CURSOS PRÁCTICOS DE:

- Visual Basic 4.0
- Programación básica
- ADA
- Y mucho más

TOWER COMMUNICATIONS, S.R.L

8 413042 303299 00030



VISUAL CAFÉ DE SYMANTEC
Nueva generación de herramientas de desarrollo Visual Java

Symantec ha anunciado la disponibilidad de *Visual Café*, basado en la arquitectura de desarrollo RAD (desarrollo de aplicaciones rápidas), lo cual asegura que las aplicaciones *Java* puedan ser desarrolladas y depuradas en una mínima cantidad de tiempo. Dicha compañía tiene el objetivo de asegurarse el liderazgo en el sector de herramientas para la Red. Como próximo paso, *Visual Café Pro* para Windows se espera que esté disponible a principios de año. Esta herramienta será un completo entorno de desarrollo de *Java* y bases de datos distribuidas.

Este pasado año la mayoría de las herramientas para *Java* estaban orientadas a desarrolladores en C/C++; *Visual Café* también es un producto apto para programadores expertos en C/C++, quienes podrán desarrollar de manera rápida sofisticados applets Java, aplicaciones y componentes. Asimismo, *Symantec Café* es la primera herramienta de desarrollo *Java* para Windows y Mac. De esta manera se satisface a un gran número de usuarios gracias a las facilidades de un entorno visual, *interfaz drag and drop*, etc. Los usuarios de *Visual Basic* o *Delphi* encontrarán en *Visual Café* como una herramienta extremadamente familiar.

El paquete en sí se puede definir como un entorno integrado de desarrollo 3GL: la versión 1.5 incorpora un nuevo compilador muy mejorado, que es ahora cinco veces más rápido que las versiones anteriores; también nuevas mejoras en el depurador, incluyendo entre otras utilidades un evaluador de expresiones.

Otras características a resaltar son los hasta cien componentes preconstruidos listos para ser utilizados.

WEBLINK 3.5 DE TEAMWARE
Nuevo visualizador Web multifuncional

TeamWARE Iberia ha anunciado el lanzamiento de *Weblink 3.5*, un visualizador *Web* que provee acceso a los servicios de la *suite* de productos *groupware TeamWARE Office 3.5*, que se ejecuta sobre *Windows NT*.

Este mismo producto puede ejecutarse en otras máquinas distintas al servidor *TeamWARE*, y permite el acceso a todos los módulos de *TeamWARE Office 3.5* a través de Internet, al utilizar el interfaz cliente universal, lo que resulta especialmente interesante para usuarios móviles y remotos.

Weblink 3.5, además de visualizar y localizar información, permite compartirla y crear nueva información y ficheros. Ofrece soporte tanto de los formatos de documentos existentes, sin necesidad de conversión al formato HTML, como de la publicación selectiva de información, utilizando productos de otros fabricantes. *Weblink 3.5* está basado en estándares de Internet como HTTP, HTML, *Java* y CGI, y permite el uso de *browsers* de Internet estándar, como *Netscape Navigator*, sobre una amplia variedad de plataformas hardware y software, incluyendo estaciones de trabajo Mac y *Unix*.

Criterios como la seguridad y la gestión a través de estándares como SSL han sido tenidos en cuenta a la hora de desarrollar este producto; también ofrece los servicios de seguridad adicionales de *TeamWARE Office*, como la verificación de nombre de usuario y la contraseña, y la sencillez de gestión que permite un control flexible de los derechos de acceso de los usuarios.

Para más información:
http ://www.teamw.com

VERSIÓN BETA DE MULTIBASE/COSMOS
Multibase, nueva generación de herramientas de desarrollo

Transtools, empresa de capital y tecnología totalmente española, presentó la versión *beta* de una nueva generación de herramientas de desarrollo *Multibase/Cosmos*, que pretenden ser una respuesta eficaz y de futuro a las nuevas necesidades de muchos usuarios.

El desarrollo de la herramienta ha costado más de dos años al departamento I+D de esta empresa. *Multibase/Cosmos* destaca especialmente por su extraordinaria sencillez de manejo y por la elevada productividad que ofrece en el desarrollo de aplicaciones. La aplicación incluye una revolucionaria herramienta de programación visual que independiza el 4GL del interface de usuario y racionaliza la tecnología basada en la orientación de objetos (posibilita la reutilización de los componentes software, y acelera y simplifica los procesos de desarrollo y mantenimiento de aplicaciones).

Para más información :
Transtools-Angel Ruiz,
angel@ttools.es

LINGUISTX SEARCH´97
Asociación entre las compañías InXight y Verity

LinguistX, *suite* que consiste en software de lenguaje natural para el análisis y recuperación de información basada en texto de la compañía californaiana *InXight*, ha sido licenciado a *Verity Inc.* Esta compañía utilizará *LinguistX* en su familia de productos *Search'97*, motor avanzado de búsqueda y recuperación.

De esta forma, ambas compañías colaborarán en el desarrollo y la mejora de estas tecnologías. El objetivo, por tanto, en esta asociación es conseguir búsquedas más exactas y rápidas con la nueva *Search'97* orientada a Internet e *Intranets* corporativas, así como cualquier aplicación software basada en texto.

Lo que es el motor de la aplicación *Search* ha sido incorporado en multitud de aplicaciones de diversas compañías incluyendo *Netscape, Lotus, Informix* y *Sybase.*

Search'97 combina su poderoso motor de búsquedas con un interfaz inteligente que puede ser integrado en muchas aplicaciones. El programa guía al usuario de una manera sencilla e intuitiva a través de la información, y provee de un simple y exacto sistema de envío que trabaja simultáneamente con múltiples fuentes.

Para más información :
http ://www.verity.com

CLARIS HOME PAGE 2.0
Disponible para MacOS y Windows 95/NT

La nueva versión del popular programa de creación de páginas *Web* para ambos entornos *Mac OS* y *Windows 95/NT*, cuyo diseño se adapta tanto a principiantes como a usuarios más avanzados es una solución WYSIWYG para crear *Web*. Ofrece la posibilidad de diseñar y desarrollar potentes páginas personalizadas en cuestión de pocos minutos sin tener que aprender el código HTML o a programar.

Asimismo, ofrece también todas las posibilidades y herramientas que necesitan los *Web masters* tales como *Frames*, tablas, librerías, soporte de *plugs-in* multimedia, publicación vía *ftp*, etc.

En cuanto a los principiantes, una de las nuevas funciones que apreciarán particularmente son las veinticinco plantillas que se entregan con el programa y que permiten crear una página personal, calendarios, presentaciones... Estas plantillas, fáciles de utilizar, incrementan la productividad ofreciendo múltiples variaciones de estilo para un total de casi cien páginas.

La versión 2.0 ofrece conformidad o "preservación de la selección" entre los modos WYSIWYG y HTML, lo que permite a los usuarios ir al sitio preciso donde estaban trabajando en el modo anterior.

Para más información :
http ://www.ntwk.es/plugin/

VISUALAGE FOR PACBASE
Para el desarrollo de aplicaciones cliente/servidor

IBM y CGI anuncian la disponibilidad de una nueva oferta tecnológica para el desarrollo de aplicaciones cliente/servidor, *Visualage for* PACBASE. Es uno de los primeros módulos que se realizan dentro del marco de la estrategia de desarrollo de aplicaciones conjuntas entre ambas empresas. El nuevo producto trata de combinar las ventajas de PACBASE de CGI, es decir, un único repositorio, enfoque *model-driven* y potentes generadores, junto con las características de *Visualage*, como, por ejemplo, orientación a objeto, construcción visual de componentes, amplia gama de librerías que proporcionan acceso a todas las tecnologías de vanguardia, como multimedia, Internet e *intranet*.

El nuevo producto genera sistemas cliente/servidor para un gran número de clientes y servidores, tanto IBM como no IBM. El producto incluye los generadores de componentes servidores, de clientes y de *middleware*.

Para más información :
GSP Imagen y Comunicación
Carlos Sánchez (91) 447 43 94

ALIAS/WAVEFRONT PARA PLAYSTATION
Las herramientas de *Alias* para la consola

Sony Computer Ent. ha seleccionado el software *PowerAnimator* como la herramienta de aplicación autorizada para la consola de juegos *PlayStation*. El acuerdo facilitará el trabajo de los desarrolladores de juegos al mejorar el proceso de creación de contenido y el entorno de desarrollo de software *Playstation*.

Como parte de este acuerdo, *Alias/Wavefront* ha desarrollado traductores de datos para exportar activos digitales directamente de *Power Animator* a la famosa consola de juegos. La estrecha integración del software de creación con el entorno de desarrollo de juegos optimiza el flujo de trabajo interactivo entre la creación de contenido y el desarrollo del juego, permitiendo así a los programadores de software para la consola disponer de más tiempo para el desarrollo de títulos creativos y menos en el proceso técnico de intercambio de datos.

Concretamente, el software de traducción *Playstation* de *Alias/Wavefront* soporta la exportación de datos de geometría, materia, texturas, iluminación y animación a la consola en formatos ASCII y binario. Sony distribuirá este software entre los desarrolladores autorizados en el CD-ROM *PlayStation Developers*, que se espera esté terminado para principios de este 97.

Alias/Wavefront es líder en el desarrollo de gráficos 2D y 3D para los profesionales de los vídeo-juegos; entre los desarrolladores de *Alias* se encuentran *Electronics Arts, Id Software, Acclaim, Virgin Interactive, Namco* y *Nintendo*.

Alias/Wavefront es filial de *Silicon Graphics*, aunque funciona de manera totalmente independiente.

Para más información :
http ://www.aw.sgi.com.

INTERNET VIDEOPHONE DE INTEL
Nuevo vídeo teléfono para la Red

Intel ha comenzado a comercializar un nuevo vídeo teléfono que permite a los usuarios verse y escucharse utilizando PCs conectados a Internet. Esto significa una nueva dimensión de las comunicaciones Internet, al permitir a los usarios hablarse y verse mientras que utilizan otras aplicaciones Internet, como navegar por el *Web*, intercambiar fotos o jugar.

El *Intel Video Phone* funciona en los PCs basados en tecnología *Pentium* con el sistema operativo Windows 95 y una cámara de vídeo. Soporta los estándares de comunicaciones internacionales H.323 e interopera con los otros software de comunicaciones basados en H.323 de *Microsoft* y de otros fabricantes. El vídeo *Phone* se coloca en el fondo del navegador de *web* del usuario y proporciona ventanas de vídeo de ambos interlocutores.

Entre otras características destacan las siguientes:

- Interfaz gráfico de fácil uso diseñado para completar las características "*point-and-click*" de los navegadores más populares.
- Listado de llamadas rápidas para mantener los números más frecuentemente utilizados.
- Una utilidad que proporciona un informe sobre la calidad y las prestaciones de una llamada Internet junto con detalles para mejorar la calidad de la llamada.
- Soporta las llamadas de sólo audio y recepción de sólo vídeo.

WINDOWS CE DE MICROSOFT

Primeros prototipos que incorporan el nuevo sistema

Microsoft Ibérica ha anunciado la disponibilidad en los Estados Unidos de los primeros dispositivos de hardware con el nuevo sistema operativo *Microsoft Windows CE* incorporado. De nuevo la plataforma de sistemas operativos Windows se amplía, ahora con *Windows CE*, diseñado para dispositivos personales móviles. Los ordenadores de bolsillo son los primeros en incorporar *Windows CE. Casio, Compaq Computer, Hewlett-Packard* y *Phillips*, entre otras, han presentado ya sus primeros modelos con *Windows CE* incorporado. Se trata de nuevos equipos que permiten trabajar con versiones especiales de las aplicaciones más habituales, como *Word, Excel* o el *Explorador* de Windows.

El interfaz de CE es fácilmente reconocible por los usuarios de Windows 95 e incluye elementos estándar como el Botón de Inicio, la Barra de Tareas y el Explorador. Asimismo, CE ofrece una arquitectura de comunicación que permite las posibilidades de comunicación tanto por cable como sin él. Integra los protocolos TCP/IP y PPP, así como las API's de comunicaciones Windows, permitiendo que los ordenadores de bolsillo puedan conectar con Internet y acceder a servidores remotos.

CE ha sido diseñado para aprovechar el conocimiento de miles de desarrolladores de software del entorno Windows. Con herramientas como *Visual C++* y una serie de *APIs Win32*, los programadores aprovechan sus conocimientos actuales para crear soluciones para la plataforma *Windows CE.*

SERVANET DE TANDEM

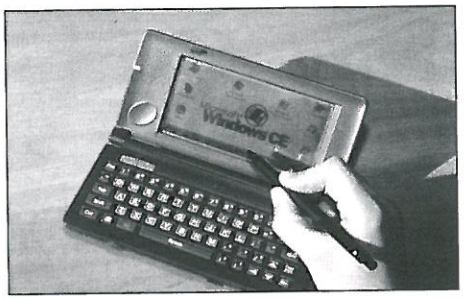
Soporta Oracle Parallel Server

Tandem anuncia su tecnología *ServerNet*, la primera que cuenta con una red de área de sistema como arquitectura principal, soportará el software de bases de datos paralelas de *Oracle Corporation, Oracle7 Parallel Server* (OPS) en entornos *Windows NT Server.*

La tecnología *Servernet*, primera solución de *luster* adaptada a SAN para grandes bases de datos paralelas que funciona sobre múltiples plataformas *Windows NT*, junto con OPS, hace posible una capacidad de crecimiento sin precedentes de las prestaciones de bases de datos en las redes empresariales.

Por otra parte, *Tandem* y *Compaq* están trabajando con *Oracle* para optimizar y probar la combinación *OPS/ServerNet/Windows NT Server.* Las tres compañías demostraron el OPS ejecutándolo sobre *Compaq ProLiant* y los servidores Series basados en *WindowsNT* de *Tandem* en la última feria *Oracle* de San Francisco.

Al ser la primera red de área de sistema disponible, la tecnología *ServerNet* se está convirtiendo rápidamente en el estándar de facto para el intercambio de datos a alta velocidad y alta disponibilidad entre sistemas de *cluster.* La arquitectura de interconexión *ServerNet* se puede configurar para proporcionar una gran capacidad de movimiento de datos. Esta inmensa



capacidad de crecimiento se consigue con pocos gastos generales de software en los sistemas de *cluster.* Esta tecnología implanta en el hardware toda la funcionalidad de comunicaciones que se necesita para garantizar la entrega de todos los paquetes de datos, en orden y sin errores.

Para más información :
firstlinepress@mad.servicom.es

SILICON GRAPHICS Y UNIVERSAL SERVER

Apoyo al SGBD de *Informix*

Silicon Graphics anuncia su apoyo a *Universal Server* de *Informix*, el nuevo sistema de gestión de bases de datos orientadas a objetos, totalmente optimizado, integrado y abierto.

Al combinar la arquitectura dinámica escalable (DSA) de *Informix* y su exclusiva tecnología abierta *Datablade* con capacidad de crecimiento ilimitada, *Universal Server* es perfectamente adaptable en entornos de computación corporativos, para desarrollar aplicaciones OLTP y de *data warehouse*, además de otras aplicaciones nuevas y emergentes que están redefiniendo los negocios y las estrategias de tecnologías de la información, entre las que se encuentran la gestión de contenidos, el *web* y el comercio electrónico.

Se podría resumir que *Universal Server* es una progresión natural e innovadora del motor de bases de datos *Informix*. Tiene capacidad para aprovechar al máximo los microprocesadores RISC y el sistema operativo IRIX, ambos de 64 bits, para mejorar las capacidades de futuras versiones.

Para más información :
firstlinepress@mad.servicom.es

NOVEDADES

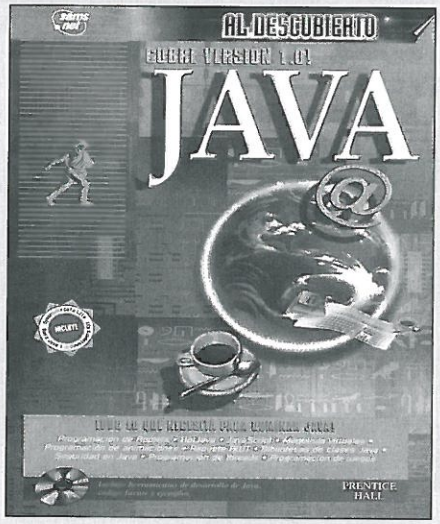
INTERSOLV DATADIRECT EXPLORER

Entorno inteligente de negocio completo

Ernst & Young distribuye en España *DataDirect Explorer*, entorno inteligente de negocio completo, que ayuda a los usuarios a convertir los datos en información significativa. *Explorer* posee varias herramientas que facilitan la recogida y análisis de datos, a destacar un constructor de consultas, un constructor de informes, un planificador automático de tareas, conexión con correos electrónicos personalizable con perfiles de usuario, etc. Es decir, incluye potentes herramientas de análisis y acceso a un gran número de bases de datos. Facilita la compartición de información y el fácil acceso a los datos mediante la herramienta *Smart Data*, la cual ayuda al usuario a comprender y usar datos corporativos mediante la ocultación de las complejas sentencias SQL.

Asimismo, *Intersolv* ha presentado *DataDirect ODBC Pack*, que permite conectar cualquier aplicación que cumpla los estándares ODBC con las principales bases de datos del mercado (más de treinta y cinco bases de datos en más de ocho diferentes sistemas operativos).

LIBROS



Java al descubierto: todo lo que necesita para dominar Java

Java es un lenguaje de programación muy reciente y sus posibilidades todavía no han sido explotadas en profundidad, pero su futuro parece resplandecer. Su utilización no se limita a la Web, su tecnología se puede aplicar a dispositivos manuales, teléfonos y vídeos, al desarrollo de bibliotecas de herramientas y aplicaciones para terceros y, con su integración en el Lenguaje Modelador de Realidad Virtual (VRML), conseguirá dotar a la Web de un nivel inimaginado de comunicación e interacción.

Java al descubierto reúne un conjunto de expertos que realizan una obra de alta calidad, lo que la convierte en referencia obligada para todas aquellas personas interesadas muy especialmente en conocer en profundidad todo lo relacionado con este lenguaje, y el conjunto de herramientas software que facilitan el desarrollo de contenidos ejecutables.

Editorial: Prentice Hall
Recomendado por Sólo Programadores
898 páginas
Idioma: Castellano
Nivel: Todos los niveles

Visual Basic 4 para programadores

La obra está especialmente indicada para aquellos programadores familiarizados con conceptos como rutinas y estructuras de programación, independientemente del lenguaje que utilicen, que requieran obtener la esencia de Visual Basic de una forma rápida y concisa. Este libro se adentra en las nuevas maneras de pensar necesarias para manejar la programación Visual Basic, radicalmente distinta de las formas de programación tradicionales, tratando temas como el modelo de orientación a objetos conducido por sucesos, el manejo de errores, las técnicas de depuración y de manejo de archivos, el intercambio dinámico de datos (DDE), el enlace y la incrustación de objetos (OLE), la automatización OLE, los gráficos, la interfaz de usuario y las bases de datos de Visual Basic 4.0.

Editorial: Osborne/McGraw-Hill
Autores: Gary Cornell y Troy Strain
342 páginas
Idioma: Castellano
Precio: Consultar editorial

TE BUSCAMOS...

SI TIENES AMPLIOS CONOCIMIENTOS O EXPERIENCIA EN ALGUNO DE ESTOS CAMPOS

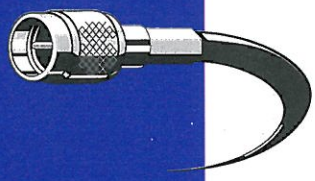
- Programación en Visual Basic, Visual C++, Delphi, HTML, JAVA, JAVAScript, CGI
  (Ref. PROG)
- Lenguajes multimedia como Director, Authorware, Toolbook
  (Ref. MULT)
- Dominio de sistemas operativos DOS, Windows 95, Windows NT, UNIX, Linux, OS2.
  (Ref. S.O)
- Redes, Conectividad y Sistemas abiertos
  (Ref. RED)
- Diseño y autoedición en CorelDraw, Photoshop, FreeHand, QuarkXPress, Page Maker
  (Ref. DIS)
- Infografía con 3D Studio, LightWave
  (Ref. INF)
- Internet, Navegación, Videoconferencia, ActiveX, Plug-Ins, Shockwave, RDSI
  (Ref. NET)
- Nuevas tecnologías, MMX, USB, DVD, Telefonía móvil
  (Ref. TECN)
- Instalación, reparación y mantenimiento de software y hardware
  (Ref. INST)
- Hardware: Tarjetas gráficas, Modems, Cámaras digitales, Scaners, Tarjetas de sonido
  (Ref. HARD)
- Inteligencia Artificial, Sistemas Expertos, Redes neuronales, Robótica
  (Ref. SE)

COLABORA CON NOSOTROS

Envíanos un currículum vitae con la referencia correspondiente a la siguiente dirección:
TOWER COMMUNICATIONS - APARTADO DE CORREOS 54.283 - 28080 MADRID

JAVASCRIPT: UNA CHISPA DE VIDA WEB

Fernando J. Echevarrieta



A lo largo del más de un año de vida de esta sección, se ha realizado una andadura por el mundo de la tecnología WWW, comenzando por una exposición del estado del arte y continuando con el estudio de las tecnologías de presentación y programación tanto en el lado del cliente como en el del servidor. Así, en forma de miniserie, se trató en un principio el lenguaje HTML en profundidad y, a continuación, antes de adentrarse en la programación CGI, la sección se detuvo un momento para tratar temas de instalación y administración de servidores WWW.

Una vez concluido el tema de la programación CGI, la sección da un nuevo giro para abordar otras tecnologías de programación y presentación de documentos. En este artículo se comienza por la presentación del lenguaje *JavaScript*.

La evolución de los estándares tiende a una convergencia de todas las tecnologías que giran en torno al www

En una publicación como ésta, dedicada a la programación, la mayor parte de los artículos desde entonces se dedicaron a la interfaz CGI, estudiando a fondo la especificación y proponiendo una metodología de programación. Durante este camino, la evolución del lenguaje HTML hizo que el tema se cruzara nuevamente en el camino, en forma de algunos artículos de actualización y, del mismo modo, el *boom Java* se hizo patente en la serie, primeramente desde un artículo teórico que estudiaba los fundamentos del lenguaje y su posible repercusión en el mercado y, más tarde, mediante la exposición de su relación con la creación de documentos HTML y la inclusión y manejo de *applets* Java desde estos documentos. Aún quedan dos tecnologías por tratar a las que, por el momento, se ha citado someramente: el lenguaje de realidad virtual en Internet, VRML, y el lenguaje de *scripts* interpretado JavaScript.

La evolución de los estándares y las estrategias de mercado tienden a establecer una relación cada vez más estrecha entre todas estas tecnologías, posibilitando ya la comunicación entre programas JavaScript y *applets* Java en un documento, así como la definición de las leyes físicas de mundos virtuales *Moving Worlds* (o VRML 2.0) mediante cualquiera de estos dos lenguajes.

VRML es un lenguaje de descripción de escenas 3D y, aunque en breve aparecerá un artículo exponiendo con detalle esta tecnología, es con la versión 2.0 cuando tiene sentido la actuación de un programador, precisamente para dotar de significado a estos mundos.

Java y JavaScript son dos lenguajes de propósito más general, que permiten el desarrollo de aplicaciones completas accesibles a través de WWW. Sin embargo, la complejidad de un lenguaje como Java es próxima a la de un lenguaje como C++, mientras que la de JavaScript es cercana a la de uno como

dBase, Clipper, o *sh*. Por otra parte, para programar JavaScript es suficiente con un editor de texto, mientras que la programación Java requiere un compilador específico. Por estas razones y aunque la importancia de Java es mayor, sus similitudes y la mayor sencillez de JavaScript hacen didácticamente interesante aprender este último lenguaje como paso previo a adentrarse en el mundo de Java, especialmente si no se conoce la metodología de programación orientada a objetos.

Así pues, con este artículo se comienza una miniserie dedicada al lenguaje JavaScript que, como es habitual, se verá interrumpida cuando sea necesario para mantener al lector al día en los avances de otras tecnologías, como las citadas de Java, HTML o VRML. Con ella, el lector podrá primeramente dotar de una interactividad a sus documentos, interactividad que antes sólo se lograba mediante aplicaciones CGI y a veces ni así era posible. Posteriormente será capaz de desarrollar aplicaciones completas a la vez que se irá familiarizando con una terminología basada en objetos y una sintaxis muy próxima a Java, que le servirán de puerta abierta a este lenguaje.

INTRODUCCIÓN

JavaScript es un lenguaje de scripts compacto basado en objetos.

- El hecho de que sea un lenguaje de scripts implica que no está diseñado para el desarrollo de grandes aplicaciones, aunque será más que suficiente para el desarrollo de aplicaciones WWW completas o para el desarrollo de interfaces WWW hacia otras aplicaciones más complejas.
- El hecho de encontrarse basado en objetos y no orientado a objetos será un punto clave a la hora de la sencillez, como se verá más adelante al compararlo con Java.

Un script realizado en JavaScript puede ser incrustado en un documento HTML proporcionando un mecanismo para la detección y tratamiento de eventos como *clicks* de ratón o validación de entradas realizadas en *forms*. De esta forma, el desarrollador de páginas HTML puede ir más allá de una mera presentación de los documentos siendo capaz de aportar significado o comportamiento a cada uno de los objetos que aparecen en sus páginas. Así, el desarrollo de páginas WWW se convierte en programación orientada a eventos y permite la realización de interfaces WWW de forma similar a como actúan herramientas de autor como *Director* o *Toolkit*.

De este modo, por ejemplo, desde JavaScript será posible detectar qué *plug-ins* tiene instalado el usuario en su *browser* y, en función de esta información, se le podrá enviar un tipo de información u otro o, simplemente, conducirle hacia la carga del *plug-in* necesario. Análogamente, se podrá detectar cuándo pasa el ratón sobre determinada zona, lo que hará posible, por ejemplo, la creación de botones animados.

Pero antes de adentrarse en el lenguaje es necesario situarlo exactamente en su lugar respecto a otros puntos de referencia conocidos por el lector: CGI, Java y HTML.

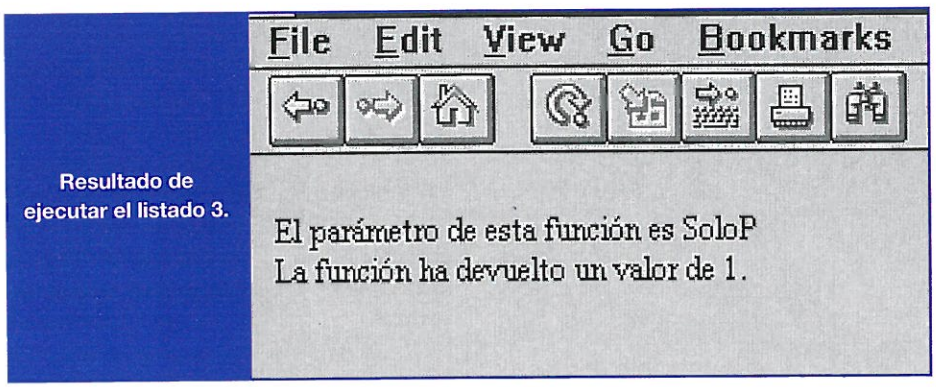
JAVASCRIPT Y CGI

JavaScript y CGI buscan un objetivo común, por lo que su campo de aplicación es muy similar; sin embargo se trata de dos aproximaciones radicalmente diferentes a un problema parecido.

Mientras que, por una parte, CGI es una interfaz entre programas de aplicación y servicios de información, JavaScript es un lenguaje de programación. En el primer caso se define un conjunto de reglas que deberán cumplir tanto el programa como el servidor, pero se deja al programador la libertad de elegir el lenguaje que considere más adecuado. En el segundo no existe ninguna restricción en el servidor, hasta tal punto que ni siquiera es necesario que éste exista. Esto permite realizar aplicaciones WWW distribuibles en disquete o CD-ROM y ejecutables en cualquier sistema sin necesidad de disponer de un servidor de WWW ni de cualquier tipo de privilegio, como ocurría con la programación CGI. Por ello, cualquier lector podrá desde el primer momento desarrollar aplicaciones JavaScript del mismo modo que realiza páginas HTML, sin necesidad de instalar ningún servidor ni de solicitar ayuda a ningún administrador del sistema.

Otra ventaja adicional que preocupa a los lectores, como reflejan en sus *mails* al autor, es la posibilidad de realizar aplicaciones que funcionen en entornos de Microsoft. Estos entornos disponen de una interfaz CGI propia diferente del estándar, por lo que para trabajar en CGI es recomendable un sistema operativo tipo UNIX. Por otra parte, en estos entornos, otras tecnologías de Microsoft han superado ya a este estándar, aunque estas tecnologías no son compatibles con ningún sistema que no sea de Microsoft. JavaScript depende únicamente del cliente, no del sistema operativo. Se trata de un lenguaje interpretado por el *browser*, por lo que las aplicaciones funcionarán en todo sistema operativo que cuente con un cliente preparado para ello.

Netscape Navigator, el primer cliente en incorporar JavaScript, se ejecuta sobre 16 plataformas, incluyendo los entornos de Microsoft y, para aquellos que prefieran el *MS Internet Explorer*, la buena noticia es que también implementa JavaScript. Por tanto, se trata de un lenguaje que, por el momento, asegura tanta compatibilidad como Java (cada uno en su campo, como se verá en el punto siguiente). La mala noticia para los seguidores del estupendo *lynx*, es que éste aún no interpreta



Resultado de ejecutar el listado 3.

JavaScript. Pero al contrario que los anteriores, es un software libre, por lo que se facilita con las fuentes y puede que pronto alguien le añada esta funcionalidad. Quizá algún lector de Sólo Programadores se anime a ello.

La única desventaja es que al trasladar toda la complejidad al cliente, la época de los *browsers* que ocupaban 300Ks y funcionaban sobre cualquier modesto ordenador ha quedado atrás.

Otra ventaja de trasladar el código al cliente es que cada evento a ser procesado no requiere comunicación a través de la red como ocurría con CGI. Esto supone un gran ahorro, sobre todo en el procesamiento de *forms*. En concreto, ahora es posible realizar filtrado de *forms* que se envían a través de *mail (URL mailto:)* sin necesidad de un CGI que evalúe y compruebe cada uno de los campos. El propio documento mediante JavaScript puede realizar estas acciones y generar respuestas e indicaciones al usuario hasta que éste haya cumplimentado correctamente el formulario. Así se garantiza que sólo se recibirá por *mail* la información deseada. Se volverá sobre ello en un artículo dedicado al procesamiento de *forms* desde JavaScript.

Por todo ello parece, en principio, que con la aparición de JavaScript la era CGI ha muerto. Nada más lejos de la realidad. Ni Java a corto/medio plazo, ni JavaScript van a desterrar a CGI del panorama WWW ya que se trata de tecnologías complementarias.

Por ejemplo, cada vez que haya que centralizar una información en un servidor será necesario realizar una comunicación con el mismo y allí seguramente estará esperando un CGI. En el caso de los *forms*, JavaScript ahorrará el trabajo del primer filtro y validación de los campos, pero una vez validada la información, a no ser que se envíe por *mail*, será necesario un script CGI en el servidor para procesarla.

Un script JavaScript puede incrustrarse en un documento HTML permitiendo la detección y tratamiento de eventos

Hoy por hoy, el acceso a bases de datos sigue encontrándose controlado por CGIs, aunque Java tendrá mucho que decir en el futuro; pero, de momento, lo más simple es disponer de CGIs en el servidor que hacen consultas a la base de datos y generan documentos dinámicamente. Estos documentos, por supuesto, podrán contener JavaScript incrustado en el código HTML, con lo que ambas tecnologías convivirán en armonía.

Y donde JavaScript no se muestra en absoluto válido es en el desarrollo de aplicaciones concurrentes y/o de acceso compartido. Imagine el lector el juego del buscaminas que se ha desarrollado en artículos anteriores. Considerando que se podría haber realizado como una aplicación JavaScript en un único documento HTML susceptible de ejecutarse localmente y que ha sido necesario disponer de conocimientos sobre arquitecturas cliente-servidor y programación con *sockets* para desarrollarlo como CGI, podría parecer que lo que se ha hecho es "matar moscas a cañonazos". Y en efecto, así ha sido. Salvo que el juego desarrollado se convierte en multiusuario simplemente con que dos usuarios introduzcan el mismo nombre y *password*, con lo que pasarán a compartir tablero (aunque convendría modificar la interfaz para que fueran conscientes de ello y evitar el caos). Desde JavaScript, esto es imposible. Por otra parte, era posible abandonar una partida y reanudarla tiempo después donde se dejó. Esta vez el estado se encuentra en el cliente y el servidor no tiene conciencia de ello. No hay que preocuparse por la programación concurrente, simplemente porque no se puede hacer. Para ello sería necesario emplear CGI.

JAVASCRIPT Y JAVA

Las diferencias entre JavaScript y Java son notables pero también las similitudes, como se verá a continuación.

Para empezar, JavaScript es un lenguaje de scripts interpretado, mientras que Java es un lenguaje compilado. En el primer artículo que se dedicó a Java en esta sección se matizó con cuidado esta última afirmación, ya que, en realidad, el desarrollador produce una compilación de clases Java que generan ficheros de *byte-codes* en el formato propio de la máquina virtual Java. Estos *byte codes* deberán ser posteriormente interpretados por esta máquina virtual que en el caso de WWW se encuentra alojada en los clientes. El caso de JavaScript es mucho más sencillo. Se trata de un lenguaje interpretado "sin reparos". El *browser* WWW que hace de cliente lee línea a línea el código JavaScript y va ejecutando.

TABLA 1

JavaScript	CGI
Es un lenguaje de programación.	Es una interfaz. Da libertad de elección del lenguaje.
No requiere un servidor WWW.	Exige la presencia de un servidor WWW.
No requiere una red, funciona en local.	Requiere comunicación en red. No funciona en local.
La aplicación reside en el cliente.	La aplicación reside en el servidor.
Requiere un cliente especial.	Sirve cualquier cliente, por simple que sea.
Pensado para aplicaciones monousuario.	Permite desarrollos de aplicaciones distribuidas, acceso concurrente y/o compartido.
Tiempo de desarrollo muy breve.	Tiempo de desarrollo medio.

Comparación entre Javascript y CGI.

Java es un lenguaje de propósito general con el que se puede desarrollar todo tipo de aplicaciones; los objetivos de JavaScript son mucho más modestos, pero, precisamente por ello, es un lenguaje mucho más sencillo. El programador Java debe disponer de una metodología de programación orientada a objetos, manejarse con clases y métodos y conocer los conceptos de herencia y jerarquías de objetos. El programador JavaScript, por el contrario, se maneja en un mundo en el que también existen objetos, pero no debe preocuparse por las clases ni la herencia de propiedades. JavaScript se basa en un modelo de instanciación de objetos muy simple y a la vez muy potente en el que no se necesitan instanciar ni extender clases. El sistema presenta al programador propiedades de alto nivel de los objetos, como su color o si son visibles o no, y las facultades de consultarlas o cambiarlas. Se dispone de una serie de objetos prediseñados y extensibles, pero no existen los conceptos de clase o herencia.

Mientras en Java existe una gran rigidez en el tipo de los datos, en JavaScript ni siquiera es necesario declarar el tipo de las variables que se reduce a tipos numéricos, *booleanos* y de cadena.

En definitiva, en JavaScript prima sobre todo la sencillez. Sin embargo, uno se para a mirar la sintaxis de JavaScript y descubre que es muy similar a la de Java, sobre todo en lo concerniente a estructuras de control. Por otra parte, al programar JavaScript se está empleando un modelo de objetos, su sintaxis y su terminología; por ello, para aquellos programadores no acostumbrados, la transición a Java es mucho menos dramática.

Además, existen importantes mecanismos de comunicación entre JavaScript y Java, lo que hace posible que desde el código JavaScript incrustado en un documento HTML se puedan consultar y alterar las propiedades de un *applet* Java. Esta facultad también se da con los *plug-ins*.

Respecto a la difusión o la portabilidad, está claro que no van a aparecer *chips* JavaScript en el futuro, ni hardwares dedicados a ello. Pero, por el momento, los mismos clientes WWW que soportan Java lo hacen también con JavaScript. Así, pese a que Java se perfila como una gran revolución que trascenderá el ámbito del WWW, dentro del mismo JavaScript dispone de igual difusión y precisa de menos requisitos. Su relación con el lenguaje HTML es mayor, ya que, si desde este lenguaje es posible incorporar *applets* Java a los documentos -como se expuso en el segundo artículo dedicado a esta tecnología-, JavaScript puede aparecer incrustado en las mismas etiquetas, en forma de atributos variables. Pero a continuación se verá cuál es el grado de relación con HTML.

TABLA 2	
JavaScript	**Java**
Interpretado (en cliente).	Compilado (en servidor).
Código integrado en HTML.	*Applets* invocados desde HTML.
Basado en objetos.	Orientado a objetos.
No es necesario declarar el tipo de las variables.	Necesario declarar los tipos.
Enlazado dinámico. Se comprueban las referencias a objetos en tiempo de ejecución.	Enlazado estático. Las referencias a objetos deben existir en tiempo de compilación.

Relación entre Javascript y Java.

JAVASCRIPT Y HTML

Los programas JavaScript aparecen incrustados en los propios documentos HTML como si de HTML se tratara. De esta forma pueden asociar eventos a objetos definidos desde HTML, a modo de enlaces, o producir atributos variables para las etiquetas HTML. Asimismo, el código JavaScript de un documento puede generar código HTML variable, como hacía un CGI.

Así pues, los programas JavaScript pueden integrarse con HTML de dos formas:

1. Como programas propiamente dichos, combinando funciones y sentencias, con el mismo aspecto que tendría el código de cualquier otro lenguaje, como Java.
2. Introduciendo manejadores de eventos JavaScript en etiquetas HTML.

A continuación se presenta cada uno de estos mecanismos:

LA ETIQUETA SCRIPT

La manera más convencional en que aparece JavaScript en un documento es en forma de programa. Para que el texto que compone el código del programa no aparezca en la pantalla del cliente, será necesario identificarlo como tal mediante la nueva etiqueta de HTML *<SCRIPT>*, de la forma:

```
<SCRIPT>
  Código en JavaScript
</SCRIPT>
```

Con vistas a uso futuro, esta etiqueta admite un parámetro que indica el lenguaje de script que se ha incrustado en el documento. Así, el siguiente código tendría el mismo efecto que el anterior:

```
<SCRIPT LANGUAGE="JavaScript">
  Código en JavaScript
</SCRIPT>
```

El listado 1 muestra lo que podría nuestro primer programa en JavaScript incrustado en un documento HTML. En él se observa cómo se ha tecleado código envuelto por la etiqueta *<SCRIPT>* que, en este caso, aparece en la cabecera del documento. Puede aparecer en cualquier parte del mismo, tanto en la cabecera como en el cuerpo aunque,

LISTADO 1

```
<HTML>

<HEAD>
<SCRIPT LANGUAGE="JavaScript">
document.write("Esto está escrito por programa")
</SCRIPT>
</HEAD>
<BODY>

Esto está escrito por HTML
</BODY>

</HTML>
```

Un primer documento en JavaScript.

como se verá más adelante, si se declaran funciones es recomendable hacerlo en la cabecera.

Nuestro primer programa se limita a escribir en pantalla el texto "Esto está escrito por programa", para lo que emplea el código *document.write*. En este código, *document* es un objeto creado por el sistema que hace referencia al propio documento y *write* es uno de los métodos que proporciona para interactuar con él. Como se puede apreciar, se trabaja con una sintaxis y terminología orientada a objetos, pero no ha sido necesario instanciar ninguna clase, declarar o construir ningún objeto. El resultado de cargar este documento en un *browser* que interprete JavaScript será la aparición de los dos textos, el escrito por JavaScript y el escrito en HTML, sin que el usuario sea consciente del proceso.

Con la versión 3.0 de *Netscape Navigator*, se amplía el lenguaje JavaScript y, con ello, se añaden las extensiones necesarias a HTML. Así, en esta versión, la etiqueta *<SCRIPT>* admite un nuevo parámetro *SRC*, por ejemplo:

```
<SCRIPT SRC="libreria.js"></SCRIPT>
```

Aunque se requiera la etiqueta de cierre, todas las sentencias que pueda haber encerradas entre ambas serán ignoradas. El objetivo de este atributo es actuar como un *"include"*, incluyendo en el documento un código JavaScript que pueda ser compartido por diversos documentos, es decir, posibilitar el uso de librerías.

Netscape recomienda que las librerías JavaScript dispongan de extensión *.js* y, para ser empleadas desde un servidor, habrá que modificar la configuración de éste para asociar el tipo MIME *application/x-javascript* a la extensión. En cualquier caso, se comunica a aquellos lectores que muestren preocupación al oír hablar de un servidor que si se va a actuar con ficheros locales no será necesario hacer nada.

MANTENIENDO LA COMPATIBILIDAD

Como se ha venido comentando a lo largo de la serie, toda extensión de HTML debe ser compatible con lo anterior. Aunque no se puede calificar a JavaScript como una extensión HTML, se debe hacer posible que si un *browser* que no "sabe" de JavaScript lee un documento con código JavaScript, el documento aparezca siendo ignorado únicamente el código JavaScript y sin presentar ningún efecto no deseado. Así pues, y dado que este tipo de *browsers* no sabrá que debe ignorar lo que aparece rodeado por las etiquetas *<SCRIPT>*, conviene ocultar el código a estos *browsers* haciéndolo aparecer como un comentario "ante sus ojos", delimitándolo entre símbolos de comentario HTML <!— y —>. Por ejemplo:

```
<SCRIPT>
<!— Ocultando a browsers antiguos
  Código delimitado por los símbolos de
comentario HTML
// Fin de la ocultación —>
</SCRIPT>
```

En el ejemplo se ve cómo la línea que contiene el símbolo de "fin de comentario" HTML, debe comenzar por //, que es el indicador de comentario en JavaScript. En caso contrario, JavaScript tratará de interpretar esa línea. También se pueden expresar comentarios encerrándolos entre /* y */.

LISTADO 2

```
<HTML>

<HEAD>
<SCRIPT>
<!— Ocultación a browsers antiguos
  document.write("Si ves esto, tu browser interpreta JavaScript")
// Fin de la ocultación —>
</SCRIPT>
</HEAD>

<BODY>
<NOSCRIPT>
Si ves esto, tu browser no incorpora la etiqueta &lt;NOSCRIPT&gt;
</NOSCRIPT>
</BODY>

</HTML>
```

Un test para comprobar si nuestro browser interpreta JavaScript.

LISTADO 3

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!— Oculto el código a los browsers que no lo entiendan
  function write_nombre(nombre) {
    document.write("El parámetro de esta función es ", nombre)
   return 1
  }
// fin de la ocultación —>
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
  document.write("La función ha devuelto un valor de ",write_nombre("SoloP"),".")
</SCRIPT>
</BODY>
</HTML>
```

Ejemplo de declaración y uso de una función en JavaScript.

Otra forma de mantener la compatibilidad, esta vez dando la posibilidad a los nuevos *browsers* de ofrecer un contenido alternativo, se logra mediante la adición a HTML de la etiqueta *<NOSCRIPT></NOSCRIPT>*. Cualquier *browser* que interprete JavaScript e incorpore esta etiqueta ignorará el código HTML encerrado por la misma, que será mostrado como contenido alternativo por otros *browsers*. En el listado 2 se puede ver un ejemplo de documento que informa de si nuestro *browser* entiende o no JavaScript empleando este método.

FUNCIONES EN JAVASCRIPT

Como en todo lenguaje de script que se precie en JavaScript es posible definir funciones. Para ello se emplea la palabra reservada *function* de la forma:

```
function <nombre_función>([parámetros])
{
 ...
 return <valor_retorno>
}
```

Por ejemplo, el listado 3 contiene la declaración de una función de la que más tarde se hará uso.

TABLA X		
Evento	**Manejador**	**Descripción**
click	*onClick*	El usuario pulsa sobre un elemento de *form* o un enlace
señalización	*onMouseOver*	El usuario coloca el ratón sobre un enlace o ancla
carga	*onLoad*	El usuario carga la página
descarga	*onUnload*	El usuario sale de la página
focalización	*onFocus*	El usuario selecciona un elemento de un *form* como entrada
desfocaliza-ción	*onBlur*	El usuario quita la selección de un elemento de un *form* como entrada
edición	*onChange*	El usuario cambia el valor de un elemento *text*, *textarea* o *select* de un *form*
selección	*onSelect*	El usuario selecciona el campo de entrada de un elemento de *form*
envío	*onSubmit*	El usuario envía un *form*

TABLA X		
Evento	**Manejador**	**Descripción**
Interrupción	*onAbort*	El usuario interrumpe la carga de una imagen
Error	*onError*	La carga de un documento o imagen produce un error
Des-señalización	*onMouseOut*	El usuario saca el ratón de un área (*client-side imagemap*) o enlace
Inicialización	*onReset*	El usuario pulsa el botón de *reset* en un *form*

Nuevo control de eventos en Navigator 3.0.

Cuando se emplean funciones conviene declararlas en la cabecera del documento; de esta forma se garantiza que todas las funciones se cargan antes de que el usuario tenga oportunidad de realizar ninguna acción que llame a una de ellas. En el listado 3 se puede apreciar también cómo es posible emplear tantas veces como se desee la etiqueta *<SCRIPT>*.

MANEJADORES DE EVENTOS

Al tratar de programar una aplicación para WWW, uno en seguida se da cuenta de que la mayoría de las acciones de programa deben ser activadas por eventos. Eventos usuales son, por ejemplo, un *click* de ratón o la focalización de un campo en un *form*. Como se mencionó anteriormente, la segunda forma de incrustación de JavaScript en documentos HTML consiste en la definición de manejadores de eventos en etiquetas.

La sintaxis general para realizar esto es:

```
<ETIQUETA
manejador_evento="Código
JavaScript">
```

donde *ETIQUETA* es cualquier etiqueta HTML que se pueda relacionar con un evento y *manejador_evento*, el evento en sí. Por ejemplo:

```
<A
HREF="http://highland.dit.upm.es:800
0"

onMouseOver="window.status='Echev
a\'s Server';return true">,
```

provoca la aparición en la zona de *status* de la ventana de la leyenda *"Echeva's server"* cuando se pasa el ratón sobre el enlace correspondiente. Este es un claro ejemplo de cómo, mediante la gestión de eventos y sin necesidad de escribir programas, a través de JavaScript es posible dotar a los documentos HTML de una gran interactividad.

La relación entre eventos y nombres de manejadores, así como su descripción, se puede observar en las tablas adjuntas. Los objetos definidos por HTML que pueden relacionarse con cada evento se describen a continuación:

- Focalización, desfocalización y edición: campos de texto, *textareas* y *selections*.
- *Clicks: buttons, radio buttons, checkboxes, submit buttons, reset buttons*, y enlaces.
- Selección: *text fields, textareas.*
- Señalización: enlaces.

A partir de la versión 3.0 de *Navigator*, *onBlur* y *onFocus* aplican también a ventanas y *framesets.*

CONCLUSIONES

Con lo expuesto en este artículo, el lector puede obtener ya una idea mas o menos clara de lo que, con poco esfuerzo, se puede hacer en JavaScript. En los próximos artículos se expondrá el mecanismo de objetos del lenguaje así como sus estructuras de control. Pero tampoco se dejarán a un lado otros temas de interés, como la carga de ficheros a través de *HTTP (HTTP File Upload)*.

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica relacionadas con el artículo, se anima al lector a que se ponga en contacto con el autor a través de:

E-mail Internet: echeva@dit.upm.es
E-mail CompuServe: 100646,2456
WWW: http://highland.dit.upm.es:8000

LENGUAJES Y HERRAMIENTAS (I)

Enrique de Alarcón

Se analiza este mes los lenguajes de programación más importantes que hay disponibles para el programador según el tipo de aplicaciones que necesite crear y las herramientas de desarrollo más importantes que existen en el mercado para cada uno de ellos tanto para DOS como para Windows y otras plataformas conocidas.

En las próximas páginas daremos "un paseo" por los principales lenguajes de programación que existen en el mercado del software.

A parte de las características de los lenguajes como tales, se explicará también a que área de desarrollo sirve cada uno y se detallarán las principales herramientas de desarrollo de que disponemos para cada uno incluyendo tanto compiladores como entornos completos visuales o no de desarrollo que pueden ser de cualquier plataforma que sea conocida, como son MS-DOS, Windows 3.11, Windows 95, Windows NT y OS/2, que poseen instalados la mayor parte de los PC mundiales.

A partir de este mes, como ayuda para los que tengan preguntas o dudas sobre este estudio o sobre anteriores artículos del mismo autor pueden hacerlo a través de la siguiente dirección e-mail Internet:_erde@canaldinamic.es .Si es posible, se contestará a los e-mails.

A continuación comenzamos esta visita turística por el mundo de la programación, como no, por el lenguaje C/C++, que es la principal capital...

LENGUAJE C/C++

Los lenguajes C y C++, son los más usados por los programadores profesionales de todo el mundo en la creación de todo tipo de aplicaciones, que van desde los vídeo-juegos, programas científicos, de diseño, etc.

El lenguaje C++ no es más que una versión del C ampliada y orientada a objetos (herencias, polimorfismos...).

Actualmente existen versiones visuales de este lenguaje de varias firmas, aunque los principales son BORLAND C++ 5.1, Microsoft Visual C++ 4.2 y Symantec C++ 7.2 y entornos clásicos de desarrollo no-visuales como son el Microsoft C++ 7.00 o el famoso Watcom C/C++ 10.6.

BORLAND C++ 5.1 DEVELOPMENT SUITE (VISUAL)

El paquete de Borland posee uno de los entornos de desarrollo más potentes de los existentes actualmente. Funciona tanto bajo DOS como en Windows y permite por ejemplo, abrir varias ventanas de un mismo archivo fuente simultáneamente, la creación de aplicaciones DOS y Windows de 16 y 32 bits, el uso de macros, barras flotantes de herramientas, entorno totalmente personalizable, uso de proyects, ayudas del API, del IDE, etc...

Como herramientas de desarrollo visual, posee el *AppExpert*, destinado a la creación de los esqueletos de programa y el *ClassExpert* para la generación de código fuente, recursos y sistema de ayuda.

En lo que a recursos se refiere, posee todos los típicos: Menús, aceleradores, cajas de diálogo, controles 3D, Tablas, etc...

El lenguaje C++ de Borland aparte de ser compatible ANSI-C, posee también muchas extensiones, como son las librerías de clases DOS, librerías de clases para Windows/Win32/Win32s. Posee además el ensamblador integrado y las extensiones estándar ANSI.

MICROSOFT VISUAL C++ 4.2

La versión Microsoft de este entorno visual de desarrollo es también de una gran potencia y dispone de depurado-

res (DOS y WINDOWS), utilidades prácticas, barras de herramientas, ayudas en línea, uso de *project* (aunque no múltiples proyectos), etc.

Como lenguaje C++, el cual es también compatible ANSI-C, posee menos librerías de clases que el paquete Borland, pero también incorpora extensiones estándar ANSI, librerías de Clases para Windows y ensamblador integrado.

WATCOM 386 C/C++ 10.6

El entorno de desarrollo Watcom, no es visual, pero debido a que se ha convertido en un estándar para la creación de vídeo-juegos y aplicaciones en general entre los programadores profesionales de todo el mundo, vamos a comentarlo.

Pese a ser un paquete más bien clásico, posee un entorno Windows muy agradable de usar, permite el uso de *projects* (múltiples), posee barra de iconos e incluye infinidad de herramientas y utilidades muy prácticas como el *Spy*, *Heap Walker*, *DDE Spy*, informadores de estados, editor de cajas de diálogo, ensamblador, etc...

Una característica muy apreciada de este entorno es que permite la creación de aplicaciones DOS 16 y 32 bits, Windows 16 y 32 bits y OS/2 , incluyendo varios tipos en cada uno de los modos, con lo cual se adapta a las necesidades de cualquier programador sin excepción alguna.

Como lenguaje de programación, resulta más eficiente en lo que a velocidad se refiere que el C++ del Borland (por ejemplo), también posee ensamblador integrado y una muy extensa librería de funciones.

Si se desea más información sobre este producto, se puede consultar el artículo "Watcom, el compilador de moda", aparecido en esta misma revista en el número 23.

C SET ++ 2.1 FOR OS/2

Este paquete de desarrollo, que en este caso está destinado al OS/2, es un entorno seudo-visual para la creación de aplicaciones C y C++ tanto de 16 como de 32 bits.

El conjunto se compone de tres elementos que son el *WorkFrame*, el *Developer's ToolKit* y el *C/C++ Tools*.

El *WorkFrame* es una utilidad que se integra en el escritorio del propio sistema operativo y que nos permite ver representaciones gráficas de las estructuras de objetos que forman las aplicaciones que estamos creando y donde cada icono representa un elemento como puede ser un fichero, una carpeta, etc.

El *Developer's Tool Kit* es una extensión al programa base anterior, que incorpora utilidades, ficheros, Librerías de extensiones y cabeceras para realizar enlaces con lenguaje REXX y otros componentes ajenos del sistema.

Las *C/C++ Tools* están formadas por las herramientas clásicas y básicas de todos los entornos de desarrollo, como son el editor, compilador y el depurador entre otras muchas incluidas.

El lenguaje C en si mismo, es compatible ANSI y además incluye también las extensiones SAA C que son de definición propia.

ENSAMBLADOR

El lenguaje ensamblador, base funcional de todos los ordenadores PC compatibles basados en Chips Intel x86, es el lenguaje de más bajo nivel existente y todos los lenguajes de alto nivel de tipo general poseen medios para poder usar rutinas diseñadas en ensamblador, ya sea mediante código in-line, linkado de librerías u otros procedimientos típicos.

Su importancia pierde relevancia en aplicaciones normales de gestión y especificas, pero es básico en la creación de aplicaciones que requieren mucha potencia de cálculo y/o funcionamiento en tiempo real, como son simuladores, vídeo-juegos, programas de calculo intensivo como Diseñadores 3D, etc... donde se usa como complemento al C/C++ y similares en la construcción de aplicaciones.

Es el único lenguaje en el que decimos que existen ensambladores para él y no compiladores, ya que en este lenguaje no se puede hablar de compilación de fuentes, sino de traducción o *ensamblaje*, lo que consiste básicamente en una conversión ASCII/binaria de los fuentes.

Existen varios ensambladores muy buenos en el mercado, como el MacroAssembler de Microsoft, el TurboAssembler de Borland y otros, que más o menos poseen características parecidas. Pese a ello, la necesidad de un ensamblador como elemento unitario cada vez es menos requerida, ya que casi todos los lenguajes de alto nivel incorporan uno integrado en el entorno de desarrollo o permiten escribir código ensamblador IN-LINE en los mismos fuentes, por lo que su necesidad se ve muy reducida en casi todos los casos.

LENGUAJE PASCAL

El lenguaje Pascal es muy conocido desde hace muchos años en el mundo de la informática y su característica más conocida es lo estructurado que es el lenguaje, el cual nos recuerda al C.

Debido precisamente a lo estructurado que es, se ha convertido durante muchos años en el lenguaje de aprendizaje escogido por las universidades de informática de todo el mundo donde es importante enseñar desde el principio a que los alumnos realicen bien las estructuras de los fuentes.

Desde la perspectiva del programador, el PASCAL es un lenguaje que no tiene ninguna ventaja frente a otros como es por ejemplo el C++, aunque posee en sus últimas versiones tantos recursos como aquél.

Como en todos los lenguajes muy difundidos, posee bastantes compiladores disponibles en el mercado y algún que otro entorno visual de desarrollo.

DELPHI

Este entorno de desarrollo visual, contiene un lenguaje de cuarta generación orientado totalmente a objetos (PASCAL) y que está pensado tanto para crear aplicaciones monousuario como del tipo cliente/servidor.

El entorno funciona en Windows y es similar en conjunto a otros paquetes visuales como son VISUAL BASIC o VISUAL C de Microsoft.

Otras características son que soporta controles VBX, pueden crearse aplicaciones que se enlacen con bases de datos como SQL, también puede usar el ODBC y el IDAPI, y genera ejecutables autoejecutables que no requieren de *run-times*.

El paquete incluye todas las herramientas normales de un entorno profesional que es editor, un compilador y un depurador, además de otras que cubren cualquier necesidad del programador, como es la edición de imágenes, acceso a bases de datos, editor de recursos, etc.

DELPHI 32 bits

La versión 32 bits para Windows 95 y Windows NT del mismo entorno de desarrollo tiene un funcionamiento <visual> prácticamente idéntico a la versión Windows 3.11 clásica, pero se han mejorado muchos aspectos internos *no-visibles* del lenguaje. Ahora se soportan los controles OCX de 32 bits, se soporta también OLE, se ha ampliado la librería de funciones disponible (que recordemos, es de 32 bits), hay más opciones de optimización de código y se ha mejorado el sistema de depurado de errores.

HERRAMIENTAS DELPHI:

Además de todo lo comentado sobre las diferentes versiones DELPHI, existen en el mercado infinidad de herramientas externas creadas por terceras compañías destinadas a ampliar las posibilidades funcionales del entorno Delphi.

Así por ejemplo, podremos ahora disponer de infinidad de ampliaciones en lo referente a VBX, OCX, librerias DDL, VCL y todo tipo de herramientas complementarias destinadas a cubrir casi cualquier área del entorno que estuviese incompleto o carente.

Nombres de algunas de estas utilidades son por ejemplo: BarWin, CodeBase, Async Professional 2.0, AccuSoft Image Format Library, Forehelp, etc...

Como se puede comprobar, las posibilidades de este entorno aumentan cada mes, lo que lo convierte en un serio candidato en caso de que quiera dedicarse a crear aplicaciones en PASCAL.

LENGUAJE B.A.S.I.C.

Se puede afirmar que es el lenguaje más popular y sus características durante muchos años lo han destinado a ser un lenguaje para ser usado simplemente como ayuda en la introducción a la programación para quienes acceden por primera vez a este mundo, se usó en los ochenta como intérprete en microordenadores de consumo masivo como eran los famosos Spectrum y Amstrad, que lo incorporaban en memoria ROM.

Ha sido ahora, con la aparición del entorno de programación Visual Basic en su versión ya 4.00, cuando las posibilidades de este renovado lenguaje se han llevado a sus límites, convirtiéndolo en un lenguaje útil para el desarrollo de aplicaciones normales que no sean muy exigentes, como son programas de gestión de bases de datos y aplicaciones tipo Cliente-Servidor lo cual viene dado gracias a la inclusión de herramientas para poder usar bases de datos de Microsoft Jet desde las propias aplicaciones.

Otras características de este BASIC, son que permite tecnología OLE, tipos de objetos reutilizables, soporta la inclusión de código específico de la plataforma donde va a funcionar, crea aplicaciones tanto 16 como 32 bits y da la posibilidad de poder usar los famosos controles Windows 95.

Para poder usar, como se ha mencionado anteriormente acceso a bases de datos, incorpora los motores Microsoft Jet 2.5 y 3.0 para la creación de aplicaciones en 16 y 32 bits respectivamente, así como la posibilidad de poder tener acceso al ODBC.

LENGUAJE REXX

Junto con el sistema operativo OS/2 de IBM, apareció un lenguaje que se transformó en estándar de la plataforma debido a que viene incluido en el propio Sistema Operativo, es de fácil aprendizaje y requiere poco tiempo para el desarrollo de aplicaciones.

Este nuevo lenguaje se llama REXX y está totalmente orientado a objetos.

Si comparamos este lenguaje con el C o C++, veremos que su velocidad de ejecución es inferior, aunque ello es ampliamente compensado porque requiere menos tiempo de desarrollo en la creación de aplicaciones.

El lenguaje REXX fue creado por Mike Cowlishaw, que lo diseñó para IBM como substitutivo al lenguaje EXEC2 de la misma firma.

Una vez que se vio la buena aceptación que tenía, IBM decidió integrarlo totalmente en su Sistema Operativo OS/2 hasta tal punto, que los fuentes REXX se pueden ejecutar desde la línea de comandos del S.O. al igual que se ejecutan fuentes BASIC en un intérprete.

A continuación presentamos los dos entornos de desarrollo más significativos disponibles de este lenguaje que son: Watcom VX-Rexx 2.1 y VisPro/Rexx Gold 2.1

WATCOM VX-REXX 2.1

El entorno de desarrollo REXX de la versión Watcom posee un generador de aplicaciones que incluye diseñador, editor y depurador, los cuales pueden funcionar paralelamente en multitarea dentro del entorno. El interfaz de desarrollo es muy avanzado, visual e intuitivo.

El entorno visual permite que podamos usar cualquier tipo de objeto y evento del *Presentation Manager*, disponiendo de todo el interfaz gráfico del S.O. para nuestras aplicaciones propias REXX.

Otra característica es que a medida que vamos creando nuestros programas visualmente, podemos ir viendo el código del programa que corresponde a lo que vamos <diseñando> con el ratón.

Otros valores añadidos del lenguaje son que es compatible DDE (Dynamic Data Exchange), puede realizar *multithreads* (multitarea dentro de la misma aplicación) y puede enlazarse con bases de datos DB/2 y Watcom SQL, además de otras funcionalidades de ampliación que lo hacen totalmente vinculable a bases de datos que tengan un controlador ODBC instalado.

Podemos decir que es un entorno de desarrollo muy completo, destinado a la creación de aplicaciones que no sean muy grandes. Por último decir que una vez se ha creado un programa REXX, éste necesita un *run-time* (un dichero DDL) que no requiere el pago de royalties para su uso y distribución.

VISPRO/REXX GOLD 2.02

Este entorno de desarrollo, que también es totalmente visual, es bastante parecido al anterior en muchos aspectos y en lo que a funcionalidades se refiere.

La mayor diferencia que posee respecto al paquete Watcom, es que el entorno no parte de un programa principal desde donde se accede a todas las funcionalidades, sino que posee un sistema propio de menús interactivos.

Otra diferencia es que con él se pueden crear programas totalmente autosuficientes que no requieren ninguna DDL que haga las veces de *run-time.*

Los programas que generemos tendrán capacidad *multithread* y en lo que a conectividad con bases de datos se refiere, más o menos tiene los mismos recursos que el entorno Watcom.

CÓMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

SÓLO PROGRAMADORES
Curso de programación de C++
Rutina para la reducción de colores
Programación de macros para Excel
Y además:
PROGRAMACION BAJO MS-DOS EN 32 BITS

SÓLO PROGRAMADORES
Entrevista al ganador Imagina 94
Cursos sobre Formatos Gráficos
PRIMER TORNEO DE PROGRAMACION MAS DE 500.000 PTAS

SÓLO PROGRAMADORES
Concurrencia en MS-DOS
Visual Basic vs Delphi
Sonido Estéreo en 16 bits
TCP/IP para LINUX
TORNEO DE PROGRAMACION DE DEMOS
250.000 Ptas. en efectivo para el ganador

SÓLO PROGRAMADORES
Hipertexto: más que un sistema
Reportaj Euskal P
El diseño de un lenguaje de programación
RAY TRACING: LA REFLEXIÓN DE LA LUZ

SÓLO PROGRAMADORES
Linux: Estructura del Nucleo
Creación de un compilador: Analizadores Sintácticos
CURSO DE PROGRAMACIÓN VISUAL C++ 4.0

Suscríbase enviando este cupón por correo o fax (91) 661.43.86, o llamando al teléfono (91) 661.42.11
Horario 9 a 14 y 15:00 a 18:00 h. de lunes a jueves y viernes de 9 a 15:00.

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....

Profesión.....Fecha de nacimiento.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº____ ____ ____ ____ Fecha de caducidad de la tarjeta..... /.....

☐ Domiciliación bancaria (rellenar código cuenta cliente).

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

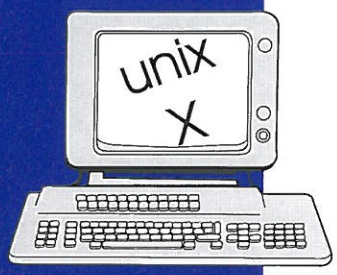
☐ Giro Postal (adjunto fotocopia del resguardo).

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28108 Pol. Ind. ALCOBENDAS (Madrid)



MENSAJES DELIMITADOS SOBRE TCP

Fernando J. Echevarrieta

Tras unos cuantos artículos dedicados a fundamentos y conceptos relacionados con comunicaciones y arquitecturas de protocolos, se retoma en este artículo el campo de la implementación, estudiando cómo proporcionar robustez al empleo de *sockets*

Hace algunos meses, el contenido de esta serie se centró en el desarrollo de aplicaciones basadas en la interfaz *socket*. En concreto, en [Echeva-1], se explicó el concepto de *socket* y se mostraron las llamadas al sistema relacionadas con la misma que el IPC (*Inter Process Communications*) de Berkeley proporcionaba. En [Echeva-2], se completaba la información explicando cómo combinar estas llamadas para realizar distintos tipos de aplicaciones cliente-servidor. Nuevamente se retomó el tema en [Echeva-3] con una introducción al desarrollo de servidores concurrentes.

En aquellos artículos la aproximación fue conceptual. Si bien, en todos se desarrollaron aplicaciones de comunicaciones siendo, de hecho, los capítulos de la serie más centrados en el código, el objetivo fundamental era mostrar las técnicas de desarrollo de aplicaciones cliente-servidor sin prestar especial importancia a los detalles de implementación. Tras haber adquirido cierto bagaje cultural durante los últimos meses en temas concernientes a redes de comunicaciones y arquitecturas de protocolos es hora, nuevamente, de descender al nivel de implementación, esta vez para cuidar todos aquellos detalles que en ocasiones anteriores se obviaron con el fin de presentar aplicaciones ejemplo didácticas y sencillas de leer. El presente artículo pretende, pues, llevar a cabo una doble función: por una parte servir como repaso de las técnicas de programación con *sockets* y, por otra, ahondar más en ellas con la finalidad de realizar programas más robustos, en este caso, mediante el empleo de mensajes delimitados sobre TCP. En cualquier caso, si a lo largo del artículo quedara algún punto oscuro, se aconseja al lector que acuda a las referencias ya publicadas en esta sección y citadas al final del mismo.

LA APLICACIÓN EJEMPLO

Con el objeto de mostrar algunos de los factores clave que será necesario tener en cuenta a la hora de programar aplicaciones de comunicaciones haciendo uso de la interfaz socket sobre TCP, se ha optado por realizar una simple aplicación. La estructura de la misma es extremadamente simple y se ha tratado de reflejar su funcionalidad en su diseño estructurado. Recuérdese que este artículo se centra en la implementación, por lo que calificar a la aplicación de "cliente-servidor" sería pretencioso, aunque, de hecho dispone de un cliente y un servidor.

Para observar algunos de los fenómenos que se comentarán, bastará con compilar el programa principal con distintos módulos que realizan diferentes implementaciones de la mismas funciones. De este modo, sin variar la estructura de la aplicación, es decir, manteniendo el diseño conceptual, se podrá atender a los detalles de implementación. El próximo mes se empleará nuevamente esta aplicación, a la que se irán enfrentando nuevas situaciones que crearán nuevos problemas que se solucionarán con nuevas implementaciones de estas funciones.

Como se mencionaba, la aplicación consta de una parte cliente, *TCPcli*, y una parte servidora, *TCPsvr*. Esta última leerá mensajes de la entrada estándar y los transmitirá al cliente. Con el objeto de realizar una serie de pruebas que justifiquen los factores que se explicarán en el artículo, se ha optado por escribir

un simple script, *prueba*, en sh que no hace otra cosa que lanzar el servidor en un puerto que toma como parámetro y redirigir a su entrada estándar las líneas que toma de otro fichero, *comandos.txt*, con un período dado. La sintaxis para realizar las pruebas será:

prueba <t> <puerto>

donde *t* es el tiempo en segundos entre mensaje y mensaje a enviar. El servidor, por su parte, deberá ser lanzado como

TCPcli <puerto>

debiendo ambos puertos coincidir. El programa de pruebas, esperará 8 segundos antes de comenzar a enviar los mensajes, por lo que habrá que lanzar el cliente en otra sesión o *xterm* en este período de tiempo.

A lo largo del artículo, se emplearán distintas funciones de envío y recepción de datos a través de *sockets* TCP desarrolladas por el autor a modo de ejemplo. Estas funciones reciben el nombre común de *sp_send* y *sp_recv* y se encuentran en librerías compuestas por los ficheros *comunica.c* y *comunica.h*. Para ello, se facilitan dos directorios *tcpnodel* y *tcpsidel*, cada uno de los cuales contiene una implementación diferente de esta librería. El módulo principal de los programas servidor y cliente permanecerá invariable en todos los ejemplos.

La forma de compilar cada ejemplo será a través de la herramienta *make*, para la que se ha preparado un fichero *Makefile* destinado al efecto. Así, bastará con teclear desde el directorio principal *tcpdelim*:

make [tcpnodel | tcpsidel]

para obtener los ejecutables *TCPsvr* y *TCPcli* correspondientes al ejemplo en cuestión.

SOFTWARE FACILITADO

Los programas de prueba se encuentran en el fichero *tcpdelim.tgz* para cuya extracción bastará teclear

tar xvfz tcpdelim.tgz

lo que generará un directorio *tcpdelim* con un subdirectorio para cada ejemplo. Los ejemplos deben ser compilados y ejecutados **desde el directorio principal** haciendo uso de la utilidad *make* como se detalla en el punto anterior. Los ejemplos han sido probados en Linux, y deberían funcionar en cualquier S.O. *UNIX POSIX*.

EL SERVIDOR TCP: TCPsvr

Antes de nada comentar que por motivos de espacio, los listados completos se encuentran en el CD de la revista. El servidor lee el primer parámetro de la línea de comando, parámetro que le indica el puerto en el que debe esperar conexiones de clientes. Tras ello, se convierte en servidor mediante el procedimiento

launch()

procedimiento que crea un *socket*, lo asocia a un puerto (*bind*) y genera una cola de espera de atención para el mismo (*listen*). Con ello, se sigue la estructura clásica de servidor TCP que se expuso en los citados artículos de la serie [Echeva-1] y [Echeva-2] y se recuerda en la figura 1.

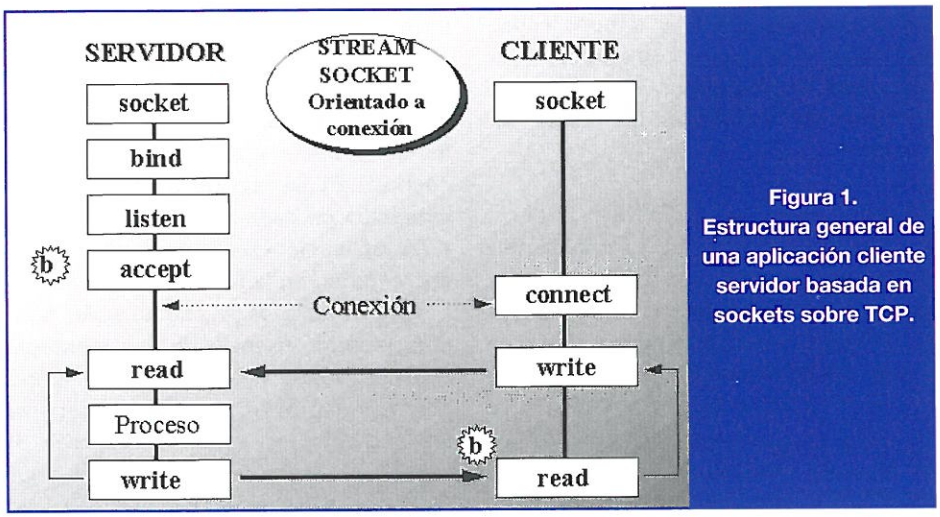


Figura 1. Estructura general de una aplicación cliente servidor basada en sockets sobre TCP.

El servidor atenderá simultáneamente al teclado y a las conexiones que pudiera recibir, por lo que se tratará de un servidor concurrente. Para implementar este tipo de servidor, tratado en profundidad en [Echeva-3], se hará uso de la llamada al sistema *select*.

Se recuerda al lector que esta llamada proporcionaba E/S asíncrona permitiendo que un proceso quedara en espera hasta que se encontrara activo un descriptor de entre un conjunto de descriptores que se le facilitaban. Y también se recuerda que los descriptores de socket y de fichero eran tratados por el sistema del mismo modo y, de hecho pertenecían al mismo conjunto. Para UNIX todo son ficheros y, por tanto, la entrada estándar dispondrá de su descriptor de fichero asociado. Así, mediante *select*, es posible "vigilar" a la vez los sockets y la entrada estándar.

Así pues, se facilita a select dos conjuntos de descriptores, por una parte *&active_sock_set*, donde se indican los descriptores que se desea vigilar y, por otra, *&sock_set*, donde *select* "apuntará" qué descriptores se han vuelto activos, es decir, si ha habido entrada por la entrada estándar o si se ha recibido algo por un socket.

Los conjuntos descriptores se manejan mediante macros. Por ejemplo, para activar la entrada estándar en el conjunto de descriptores "a vigilar" se indica:

FD_SET(STDIN, &active_sock_set)

Si no recuerda ésto y le supone un problema para comprender el código, consulte los artículos anteriores o la función *select* en el manual en línea.

Tras la llamada a *select*, comienza el bucle típico en el que se actúa en consecuencia en caso de que el descriptor que se encuentre primero activo sea el de entrada estándar, el de atención o un socket esclavo (para lo que se recorren todos los descriptores candidatos en un bucle). En estos casos, se envía el mensaje recibido a todos los clientes (*stdin_process*), se genera un nuevo

LISTADO 1 SERVIDOR TCPsvr

```
/* TCPsvr.c - Ejemplo de servidor TCP */

/*-----------------------------------------------------------------
Modulo: TCPsvr.c
Fecha: Diciembre 1996
Autor: Fernando J. Echevarrieta Fernandez
Version: 0.1
Descripción: Programa de pruebas.
Tiene un select que selecciona si le estan hablando por el socket o
por el teclado. Si le hablan por el teclado, envia un mensaje por el
socket. Si le hablan por el socket, pinta en pantalla.
argv[1]: Puerto de atención
Este programa se distribuye bajo licencia GPL
-----------------------------------------------------------------*/

#include <unistd.h>     /* fork */
#include <stdlib.h>     /* malloc */
#include <stdio.h>      /* printf */
#include <sys/types.h>
#include <sys/socket.h> /* socket, bind, listen */
#include <netinet/in.h> /* struct sockaddr_in */
#include <arpa/inet.h>
#include <string.h>  /*  strcmp,  strcp,  strcpy,  memcpy */
#include <signal.h>
#include <sys/time.h>   /* Macros FD */
#include "comunica.h"   /* sp_send, sp_recv */
#define MAXLENMSG  256
#define STDIN  0
#define STDOUT 1

char message[MAXLENMSG];
int s_atencion, ns, statuslen;
int file_desc, num_sock_set;
struct sockaddr_in i_status;

int SERVER_PORT;
int create_slave_sock(int s);
int stdin_process(void);
int procesa_esclavo(int socket);
int launch(void);

/* Variables de conjunto para el select */
fd_set sock_set, active_sock_set;

int main(int c, char *argv[])
{
        SERVER_PORT = atoi(argv[1]);

        /* Starts communication processes */
        s_atencion = launch();

printf("Lanzo el servidor\n");

      num_sock_set = getdtablesize();
      FD_ZERO(&active_sock_set);

      FD_SET(STDIN, &active_sock_set);
      FD_SET(s_atencion, &active_sock_set);

      while (1) {
            bcopy((char *)&active_sock_set,(char *)&sock_set,sizeof(sock_set));

            select(num_sock_set,&sock_set,(fd_set *)0,(fd_set *)0,
              (struct timeval *)0);

        if (FD_ISSET(s_atencion, &sock_set)) {
            ns=create_slave_sock(s_atencion);
                printf("Abierto socket esclavo [%d]\n",ns);
                FD_SET(ns, &active_sock_set);
        }

        if (FD_ISSET(STDIN,  &sock_set)) {
          stdin_process();
        }

                for (file_desc=3; file_desc < num_sock_set; ++file_desc)
                  if ((file_desc != s_atencion) && FD_ISSET(file_desc,&sock_set)) {
                    if (procesa_esclavo(file_desc) == 0) {
                        close(file_desc);
                        printf("Cerrado socket esclavo [%d]\n",file_desc);
                        FD_CLR(file_desc, &active_sock_set);
                    }
                  }

      }

}

/*-----------------------------------------------------------------*/
```

socket esclavo (*create_slave_sock*) o se escribe en pantalla lo recibido (*procesa_esclavo*), respectivamente. La información sobre cuales son los descriptores activos es devuelta por *select* en *&sock_set*.

El procedimiento *create_slave_sock*, queda a la espera de una conexión (*accept*), figura 1, a la que asociará un *socket* esclavo.

Este procedimiento, se encargará también de realizar eco del primer mensaje que reciba, como confirmación al cliente de que todo va bien. Aunque el código es claro, si el lector desea profundizar, este mecanismo también fue detallado en [Echeva-2] y repasado posteriormente en [Echeva-3].

Para retransmitir los mensajes a los clientes, el procedimiento *stdin_process*, emplea la función *sp_send*, que se implementará de diferente forma en cada ejemplo.

El bucle continuará indefinidamente, hasta que "se acabe" la entrada estándar. Cuando el servidor es invocado por el programa de prueba, esto ocurre al haberse enviado todas las líneas del fichero.

EL CLIENTE TCP: TCPcli

La estructura del cliente es más simple que la del servidor y se basa, también, en la figura 1. Tras leer el puerto al que conectarse como parámetro, crea un socket (*socket*) y trata de conectarlo a ese puerto (*connect*) de la máquina local (127.0.0.1). Una vez establecido contacto, envía un mensaje y espera su eco, tras lo que comienza un bucle de espera de datos a través del *socket*, para lo que emplea la función *sp_recv*, que también se implementará de diferente forma en cada ejemplo.

Cada mensaje recibido es escrito en pantalla. Con objeto de simular un programa más complejo, que realizará un determinado proceso tras la recepción de cada mensaje, el programa queda a la espera durante un segundo tras mostrar el mensaje recibido y antes de tratar de leer nuevamente del socket. Para ello, emplea la función *sleep()*, que suspende el proceso durante un período de tiempo especificado en segundos.

En caso de que se produzca un error, *sp_recv* devolverá un valor negativo y si la conexión es terminada, devolverá un valor 0. En cualquiera de los dos casos, el programa termina.

PARECE QUE FUNCIONA, PERO NO FUNCIONA

La primera implementación de las funciones *sp_send* y *sp_recv* es un mapeo directo sobre las llamadas al sistema *read* y *write* respectivamente, es decir, equivale a emplear estas funciones directamente, que es lo que se hizo en anteriores artículos.

Si se realiza la primera prueba, compilando con *make tcpnodel* y ejecutando *prueba 2 1400*, en una sesión y *TCPcli 1400*, en otra, por ejemplo, se obtendrá un resultado como el que se puede apreciar en la figura 2. Todo se habrá desarrollado perfectamente, "como era de esperar". Se han usado implementaciones como ésta sin problemas en artículos anteriores, como, por ejemplo, la transmisión de tablero del juego del buscaminas en red [Echeva-2] y "todo ha ido perfectamente". Pero claro, "todo ha ido perfectamente" porque se trataba de aplicaciones "con truco". Para empezar, se conocía de antemano la longitud de los datos a leer y se actuaba en consecuencia. El tiempo de proceso tras la recepción de cada mensaje era mínimo y las aplicaciones cliente y servidor se encontraban en la misma máquina.

En este caso, la prueba ha salido bien porque entre el envío de dos mensajes consecutivos al cliente le sobra tiempo para leer el primero, procesarlo (esperar un segundo) y volver a esperar al siguiente. Pero supóngase que en lugar de llegar lentamente los mensajes, lo

hacen con una frecuencia desconocida, que puede ser a una gran velocidad. Y supóngase que no se conoce su longitud (porque estos mensajes se correspondan con casillas de un tablero, por ejemplo), sino que, por ejemplo, se trata de mensajes de longitud variable que forman parte de un protocolo, mensajes que llegan a velocidad vertiginosa... Para estudiar cómo reaccionaría el programa, bastará ejecutarlo de nuevo lanzando el servidor con *prueba 0 1400* en una sesión y *TCPcli 1400* en otra. Ahora el tiempo de espera entre envío y envío será de 0 segundos, es decir, se realizará tan rápido como pueda el ordenador. Si se realiza esta prueba se obtendrá un resultado similar al que aparece en la figura 3. En efecto, en destino, los mensajes aparecen entremezclados. Ni se ha recibido el mismo número de mensajes que se han enviado ni se ha conservado su longitud. Algo ha ocurrido.

MENSAJES DELIMITADOS CON TCP

Para comprender lo que ha pasado bastará repasar por encima el artículo del número 28, dedicado a TCP. Pero para ahorrar trabajo, se recordará al lector que en él se decía que TCP es un protocolo fiable, orientado a conexión, que transmite un *flujo de bits* entre origen y destino. TCP garantiza que no hay pérdidas, ni duplicados, ni cambios de orden. Exactamente lo que se introduce por un *socket* en origen sale bit a bit por su homólogo en destino pero, ¿quién ha hablado de mensajes? No en TCP. Precisamente, uno de los factores importantes a considerar era que TCP no garantizaba la delimitación de SDUs (Service Data Unit, unidad de servicio del nivel superior, en este caso, las unidades que se envían por el *socket*), es más, podía realizar concatenación de SDUs. La interfaz *socket* tampoco se encarga de garantizar esta delimitación sobre TCP. Por tanto, será responsabilidad de las aplicaciones. En la última prueba, lo que ha ocurrido es que mientras se procesaba el primer conjunto de bits que se extrajo del *socket*, han seguido llegando más y más correspondientes a las distintas líneas del fichero y, unos tras otros, se han ido agrupando en destino esperando ser recogidos. Cuando el programa, algo más de un segundo más tarde ha vuelto a recoger lo que esperaba fuera el segundo mensaje del *socket*, ha recogido todo lo que había acumulado y le cabía en el buffer, produciendo la consiguiente mezcla de mensajes, mensajes que nunca han existido porque no se han programado como tales y TCP no entiende de mensajes.

Por ello, surge la necesidad de reprogramar las funciones *sp_send* y *sp_recv* para que sean capaces de enviar y recibir, respectivamente, mensajes delimitados empleando *sockets* TCP y ello, de manera transparente al programa de aplicación, es decir, sin cambiar su interfaz.

LISTADO 2 EL CLIENTE TCPcli

```
/* TCPcli.c - Cliente de TCPsvr */

/*-------------------------------------------------------------------
Modulo: TCPcli.c
Fecha: Diciembre 1996
Autor: Fernando J. Echevarrieta Fernandez
Version: 0.1
Copyright (c) Fernando J. Echevarrieta 1996 echeva@dit.upm.es
Este programa se distribuye bajo licencia GPL
-------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include "comunica.h"

int port;
char buf[256];

/* ------------------------------------------------ */

int main(int *argc, char **argv)
{
        int s, c;
        struct sockaddr_in mineserver;

        port = atoi(argv[1]);

        mineserver.sin_family   =AF_INET;
        mineserver.sin_port     =htons(port);
        mineserver.sin_addr.s_addr=inet_addr("127.0.0.1");

        s=socket(AF_INET,SOCK_STREAM,0);

        /* Pide conexion al server */
        if (connect(s,(struct sockaddr *)&mineserver,sizeof(mineserver)) == -1)
        {
                perror("client:connect");
                exit(1);
        }

        write(s,"Soy un cliente",sizeof("Soy un cliente"));

        read(s,buf,sizeof("Soy un cliente"));
        printf("El servidor me ha contestado'%s'\n",buf);

        while (1) {
          if ((c=sp_recv(s,buf)) > 0) {
            write(1,buf,c);
            write(1,"\n",1);
          }
          else if ( c == 0 ) {
            close(s);
            exit(0);
          }
          else {
            printf("Ha habido un error de lectura, me muero\n");
            exit(1);
          }
          sleep(1);
        }

}

/* ------------------------------------------------ */
```

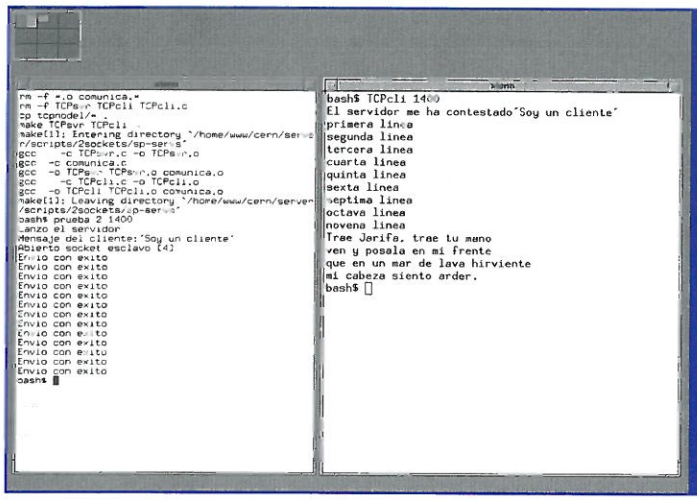


Figura 2. Resultado de la ejecución del ejemplo tcpnodel con un intervalo entre mensajes de 2 segundos.

Así, lo primero a realizar es definir un formato para los mensajes. Dado que estos podrán ser de longitud variable, lo lógico será dividirlos en cabecera y cuerpo, facilitando en la primera, información sobre el segundo y quizá cierta información redundante o de control.

En los ejemplos, se ha implementado el formato de mensajes *dp_RDO* que se emplea en alguna implementación de *RPCs* sobre TCL. El formato de estos mensajes puede observarse en la figura 4. En ella, se puede apreciar cómo cada cabecera comienza con un *número mágico*, al que sigue un campo con la longitud del mensaje completo (cabecera más cuerpo), otro campo separador y el mensaje propiamente dicho. Este formato se ha implementado mediante la estructura *msg*:

```
struct msg {
        unsigned long int id;
        unsigned long int len;
        char sep[2];
        char *message;
};
```

El empleo del número mágico es una técnica para detectar anomalías en la comunicación. Aunque se crea que el software es correcto y el sistema no produce errores, conviene comprobar que todo va bien. Así, al leer un mensaje bastará con comprobar que el *número mágico* no es el esperado para detectar la anomalía y proceder convenientemente, casi siempre reiniciando la conexión. El valor del número mágico es indiferente ya que sirve únicamente como "sello notarial". En el caso del ejemplo se emplea el valor *0x6FEEDDCC*.

De esta forma, la función *sp_read*, realiza dos llamadas *read* al sistema. En la primera lee del *socket* tantos bytes como corresponden a la longitud de una cabecera. Tras analizarla, podrá deducir la longitud del cuerpo del mensaje que será exactamente lo que tratará de leer en la segunda. La implementación de *sp_send*, por el contrario, no requerirá más que una llamada a *write*. La función construirá el mensaje completo con cabecera y cuerpo y lo enviará de una sola vez. La librería quedaría ahora tal y como figura en el listado 3 y funcionará correctamente, cualesquiera que sean sus condiciones de uso, ya que se han tenido en cuenta todos los factores... ¿o no?

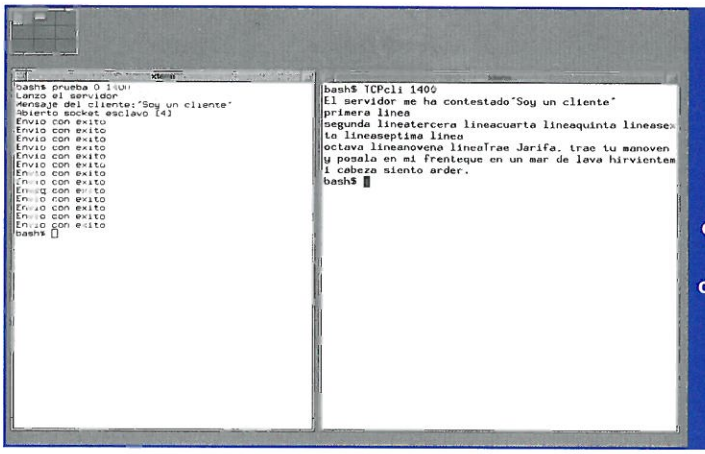

bash$ TCPcli 1400
El servidor me ha contestado"Soy un cliente"
primera linea
segunda lineatercera lineacuarta lineaquinta lineasex
ta lineaseptima linea
octava lineanovena lineaTrae Jarifa, trae tu manoven
y posala en mi frenteque en un mar de lava hirvientem
i cabeza siento arder.
bash$

Figura 3. Resultado de la ejecución del ejemplo tcpnodel con envío continuo de mensajes.

ALGO SE HA QUEDADO EN EL TINTERO

Si se prueba el ejemplo propuesto, la probabilidad de que falle es prácticamente nula, sin embargo, seguirá siendo incorrecto. Hasta ahora, se ha considerado la posibilidad de que se lea de un *socket* más información de la que correspondería a un mensaje, para lo que se ha ideado una forma de delimitar los mismos. Sin embargo, no se ha estudiado la posibilidad de que ocurra el efecto contrario, es decir, que se lea menos de lo esperado. Esto, sería mucho más probable si la prueba, en lugar de hacerse con las aplicaciones cliente y servidora en la misma máquina, se realizara con ambas en máquinas distintas.

En efecto, es bastante probable que, debido a la red intermedia, cuando se trate de leer cierta cantidad de un mensaje, no se encuentre aún toda disponible. Imagine el lector, por ejemplo, un mensaje cuya cabecera indica que el cuerpo es de 10Ks. Probablemente, al hacer la prueba en local e intentar leer 10Ks, todo iría bien, pero si la información tuviera que atravesar varias máquinas, *routers* y, quizá redes enteras, es probable que una llamada a *read* retornara un número de bytes inferior al que se ha tratado de leer sin que esto implicara ningún error. Simplemente habrá que leer el mensaje con sucesivos *read* sobre el *socket* hasta que se haya recuperado toda la información que se busca.

Para ello, se ha modificado la parte correspondiente a la lectura del mensaje de la siguiente manera:

- Se introduce un contador *totRead*, que se inicializa a cero e indicará el número de bytes leídos en cada momento.
- Se emplea una variable *tryToread* que reflejará la cantidad en bytes que se intenta leer en cada llamada a read y se inicializa con un valor igual a la longitd del mensaje esperado *msg_len*.
- En cada lectura se comprueba si se ha leído el mensaje completo y, si no es así, se actualizan los valores de ambos

LISTADO 3 FUNCIONES NO ROBUSTAS PARA COMUNICACIÓN DE MENSAJES DELIMITADOS

```
/* comunica.c Ver 0.1a : funcs to send delimited TCP messages */
/* TCP messages format adjust to RDO TCL/TK RPC messages */
/* Copyright (c) Fernando J. Echevarrieta 1996 echeva@dit.upm.es */
/* Este programa se distribuye bajo licencia GPL */

#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <netinet/in.h>
#include <assert.h>
#define INC 1
#define MAXLENMSG 256

struct msg {
        unsigned long int id;
        unsigned long int len;
        char sep[2];
        char *message;
};

/*-------------------------------------------------------------*/

int sp_recv(int s, char *buffer)
/* Strips headers from packet received */
{
        int length = sizeof(unsigned long int);
        int msg_len, totRead, tryToread;

        struct msg r_mesg;
        char *message;
        int retcode;

        message=(char *)malloc(2*length+2);
        assert(message);

        retcode = read(s,message,2*length+2);
        if (retcode <= 0) return retcode;

        /* Creates structure from buffer */
        memcpy(&r_mesg.id, message, length);
        memcpy(&r_mesg.len,message+length, length);
        memcpy( r_mesg.sep,message+2*length,2);

        r_mesg.id =ntohl(r_mesg.id);
        r_mesg.len=ntohl(r_mesg.len);

        if (r_mesg.id != 0x6FEEDDCC) {
                printf(message);
                memcpy(buffer, message, retcode);
                return retcode;
        }

        msg_len=r_mesg.len-2*length-2;

        r_mesg.message = (char *) malloc(msg_len);
        assert(r_mesg.message);

        retcode = read(s,r_mesg.message+totRead,tryToread);

        if (retcode <= 0) return retcode;

        memcpy(buffer, r_mesg.message, msg_len);

        free(message);
        free(r_mesg.message);

        return retcode;

} /* end sp_read */

/*-------------------------------------------------------------*/

int sp_send(int s,char *buffer, int len)
/* Sends content of buffer with headers added */
{
        int length = sizeof(unsigned long int);

        struct msg s_mesg;
        char *message;
        int msg_len, retcode;

        msg_len = (2*length+2*sizeof(char)+len);

        /* Defines packet fields */
        s_mesg.id      = htonl(0x6FEEDDCC);
        s_mesg.len     = htonl(msg_len);
        s_mesg.sep[0]  = 'd';
        s_mesg.sep[1]  = ' ';
        s_mesg.message = (char *) malloc(len);
        memcpy(s_mesg.message, buffer, len);

        /* Puts the structure in buffer message */
        message = (char *)malloc(msg_len);
        memcpy(message,        &s_mesg.id,  length);
        memcpy(message+length,     &s_mesg.len, length);
        memcpy(message+2*length,  s_mesg.sep, 2);
        memcpy(message+2*length+2,s_mesg.message,len);

        /* Sends data through socket */
        retcode = write(s,message,msg_len);

        if (retcode != msg_len) retcode=-1;

        free(message);
        free(s_mesg.message);

        return retcode;

} /* end sp_send */

/*-------------------------------------------------------------*/
```

contadores y se efectúa una nueva lectura.

```
do {
  retcode =
read(s,r_mesg.message+totRead,tryToread);
  if ( retcode>= 0) {
    totRead  += retcode;
    tryToread-= retcode;
  }
} while ((retcode> 0) && (totRead< msg_len));
if (retcode <= 0) return retcode;
```

El código correspondiente a la librería *comunica* con manejo de mensajes delimitados y teniendo en cuenta estas consideraciones, se encuentra en el subdirectorio *tcpsidel*. Para obtener los ejecutables, bastará compilar con *make tcppsidel*. Si se realiza de nuevo la prueba que falló anteriormente, *prueba 0 1400*, todo irá sobre ruedas.

CONCLUSIONES

Los ejemplos facilitados aún distan de ser robustos. Hay aún cabos sin atar. Por ejemplo, en *sp_read*, se ha asumido que la cabecera del mensaje, por tener un tamaño reducido sí será leída en un solo *read*, y no se ha protegido del mismo modo que la lectura del cuerpo del mensaje. También se asume que el simple protocolo de eco que se realiza al establecer la conexión es inmune a estos errores. La probabilidad de que esto no suceda así es "despreciable". Es "despreciable" en un programa ejemplo, o en juego distribuido, quizá, pero desde luego no lo es en una central nuclear. En cualquier caso, siempre que el código forme parte de una librería, de un bloque opaco o pueda ser empleado por terceros, conviene realizarlo bien. Habrá que considerar todas las posibilidades y analizar todos y cada uno de los códigos de retorno de cada llamada al sistema. Por ejemplo, no se han estudiado las llamadas a *write*, pero también podría darse el caso en que los buffers del sistema operativo se encontraran llenos y hubiera que enviar un mensaje con sucesivos *write*. El código sería análogo al que se ha realizado para el *read*.

En la mayoría de los casos, como la programación del buscaminas en red, esto no será necesario y si falla una partida de cada 10.000 no importará mucho. Pero lo peligroso no es hacer las cosas mal, si se es consciente de ello, sino creer que se hacen bien.

CONTACTAR CON EL AUTOR

Para cualquier duda, consulta, sugerencia o crítica, relacionada con el artículo, se anima al lector a que se ponga en contacto con el autor a través de:

E-mail Internet: echeva@dit.upm.es
E-mail Compuserve: 100646,2456
WWW: http://highland.dit.upm.es:8000

REFERENCIAS

[Echeva-1]
"Arquitecturas cliente-servidor", núm 17
[Echeva-2]
"Arquitecturas cliente-servidor", núm 18
[Echeva-3]
"Servidores Concurrentes", núm 21.

CICS

José María Peco

El lector debe remontarse a los años 60 para conocer las necesidades que se planteaban en aquella época y los recursos de que se disponía. Así, informáticamente hablando, sólo había procesos *batch*.

Pero una compañía aérea, concretamente la TWA, planteó un problema que no podía resolverse con los elementos existentes, razón por la que hubo que crear unas herramientas o productos nuevos, entre los que se encontró el CICS.

Para conocer la importancia de este tema, es necesario remontarse a finales de los años 50, ya que es en aquellos momentos en los que aparece un nuevo periférico: El *disco magnético*.

No se llamó '*disco*', sino **DAAD** (dispositivo de almacenamiento de acceso directo), pues previamente se había creado otro tipo de discos y tambores magnéticos que recibían esta denominación. No obstante, a pesar de disponer de este nuevo periférico, durante algún tiempo la filosofía de las aplicaciones siguió siendo secuencial, y los DAAD sólo se empleaban para contener las bibliotecas del sistema y como ficheros temporales de datos.

Así estaba el panorama informático de la época, cuando aparecieron unos clientes con unas necesidades de procesos nuevos, concretamente las líneas aéreas americanas (entre las que se encontraba la TWA). Estas compañías demandaban procesos ON-LINE, pues básicamente tenían tres tipos de transacciones básicas:

- Disponibilidad de plazas.
- Venta de plazas.
- Cancelación de reservas.

Por lo tanto, los requisitos básicos que necesitaba la compañía aérea eran:

- Un fichero maestro de plazas, actualizado al instante.
- Una buena velocidad de respuesta.

Y, como el hardware y el software de la época no permitían esto, aunque había periféricos que sí, la solución que se adoptó consistió en desarrollar un hardware y un software nuevo.

Las características solicitadas al nuevo hardware eran:

1. CPU mas rápida.
2. Desarrollo de periféricos nuevos (Terminales).
3. Desarrollo de todo un sistema de comunicaciones que pudiese enlazar los terminales con el ordenador. La infraestructura de las comunicaciones ya existía, pero eran comunicaciones entre terminal y terminal, no entre CPU y CPU.

La solución dada al software fue más revolucionaria, pues se buscó un programa de control que atendiera simultáneamente el máximo número de transacciones posibles. Se denominó : "*proceso concurrente de transacciones*".

La figura 1 muestra de forma comparativa las instalaciones anteriores y posteriores a este momento. En la **UC** (Unidad de Control) se encuentra el interfaz universal que hace posible la comunicación entre el ordenador y la impresora. Mientras que la **TCU** (*Transmision Control Unit*) es la unidad de control a la que llegan las líneas telefónicas que, en su otro extremo, tienen conectados los terminales que permiten interactuar con el ordenador central.

Pues bien, la idea básica de esta innovación consistía en tener residente en el ordenador un programa maestro cuya función fuera detectar, encolar y tratar las transacciones que llegaran desde los distintos terminales de la red; y, en función del código de la transacción recibida, cargar en memoria, junto a él, el programa de contabilidad, reserva de vuelo, etc., que permitiera tratarla.

De esta forma, se pretendía conseguir que los tiempos libres que deja la CPU al seleccionar una transacción se pudieran aprovechar para el proceso de una nueva transacción, o de una vieja retenida temporalmente.

Es de notar que en los procesos ON-LINE se fija un tope o umbral en tiempo de respuesta ante una transacción, no importando la cantidad de información procesada, a fin de impedir que

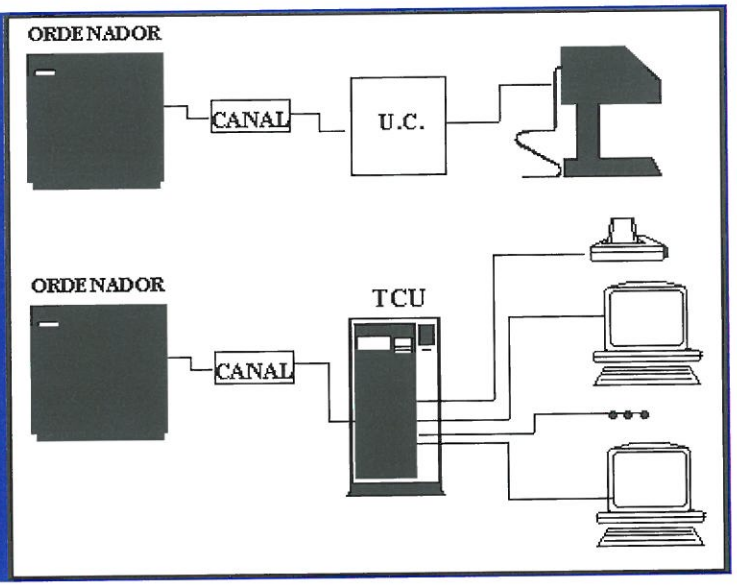


Figura 1: Instalación típica anterior y posterior a la aparición de los procesos ON-LINE.

una transacción entre en el '*abrazo de la muerte*' y se adueñe de todo el sistema.

La figura 2 representa esquemáticamente la interrelación entre la CPU al ejecutar el programa de aplicación y el **canal**, encargado de realizar las operaciones de entrada/salida. Es importante resaltar la figura del **canal**, ya que la ejecución de sus instrucciones no las realiza el procesador central; por eso, mientras se procesa el programa de canal, la CPU puede dedicarse a ejecutar otras instrucciones de otra región o de otra tarea que estuviera en espera.

INTRODUCCIÓN

CICS es el acrónimo de *Customer Information Control System* o **S**istema de **C**ontrol de **I**nformación de **C**lientes.

IBM define este producto como un sistema de Base de datos/Comunicación de uso general, pues ofrece soporte para sistemas ON-LINE de forma similar a como lo hace el sistema operativo. Pero esto no debe hacernos creer que le puede sustituir, pues se ejecuta bajo el control del sistema operativo MVS (antes DOS/VS) junto con el resto de aplicaciones de la instalación, incluidos los procesos por lotes (*Batch*).

La figura 3 presenta las distintas particiones o espacios de direcciones (*address space*) en los que se encuentra organizado el sistema central. Así, de forma muy esquemática, se puede decir que las posiciones más bajas de memoria se encuentran ocupadas por el sistema operativo, el cual se encarga de dar servicio al resto de las regiones en las que se encuentra dividido el sistema. Cada una de estas regiones constituye un subsistema, de modo que en una puede correr el CICS, en otra DB2, en otra un núcleo ADABAS, etc...

De esta forma, cuando CICS quiere un dato de un fichero, si lo tiene asignado a su entorno, invoca al Sistema Operativo para que sea éste quien llame al canal y obtenga una copia de dicho registro en memoria. Pero, si quiere un dato de ADABAS, le pide al S.O. que le pida a ADABAS dicho dato, siendo ADABAS quien se encarga de llamar al S.O. para leer dicho dato de los ficheros físicos en los que guarda la información por él gestionada, y así sucesivamente.

Otro componente que debe ser mencionado para conocer el funcionamiento de este producto, es el **VTAM** (*Virtual Telecommunications Access Method*), componente encargado de proporcionar los servicios necesarios para el control de transmisión: acepta las peticiones de inicio de comunicación, solicita la conexión de terminales para la iniciación automática de tareas, etc. En una palabra, se podría decir que es el responsable de las comunicaciones. Se ejecuta en el procesador principal, en un espacio de direcciones propio, y no debe confundirse con el **NCP** (*Network Control Program*), que es el programa encargado igualmente de las comunicaciones, pero que se ejecuta en el controlador de comunicaciones.

El VTAM permite combinar dispositivos por lotes e interactivos en la misma línea de transmisión, sin degradación de la respuesta interactiva. Los recursos del VTAM no están sujetos a limitaciones de tiempos para el programador de aplicaciones.

Una característica importante de CICS es que el diseñador del sistema no tiene que tener en cuenta las características del terminal desde el que se va a ejecutar la aplicación, punto este muy importante en la época inicial, aunque hoy, por evidente, casi impensable.

PROCESOS BATCH Y PROCESOS ON-LINE

Es curioso ver cómo evolucionan los medios. Tal y como se ha comentado en los puntos anteriores, en sus comienzos sólo había procesos batch o procesos por lotes, donde el ordenador se dedicaba íntegramente a un único programa, el cual trataba los datos, que solían ser ficheros voluminosos, y generaban listados también enormes que debían ser distribuidos a todas las oficinas, pues era el medio de consulta habitual de los resultados de estos procesos.

Hoy, evidentemente, siguen existiendo los procesos anteriores, pero los resultados ya no se presentan en papel, a no ser por petición expresa, sino que son consultados desde el propio terminal.

Por otra parte, las grandes instalaciones tienden a planificar la ejecución de

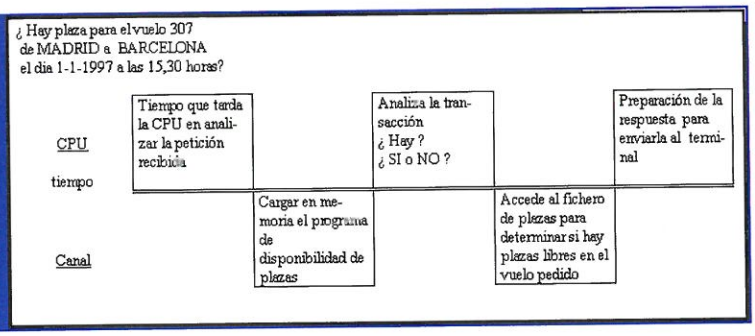

¿ Hay plaza para el vuelo 307
de MADRID a BARCELONA
el dia 1-1-1997 a las 15,30 horas?
CPU
tiempo
Canal
Tiempo que tarda la CPU en analizar la petición recibida
Analiza la transacción
¿ Hay ?
¿ SI o NO ?
Preparación de la respuesta para enviarla al terminal
Cargar en memoria el programa de disponibilidad de plazas
Accede al fichero de plazas para determinar si hay plazas libres en el vuelo pedido

Figura 2: Esquema del reparto de trabajo ejecutado por la CPU y el canal para la ejecución de una transacción.

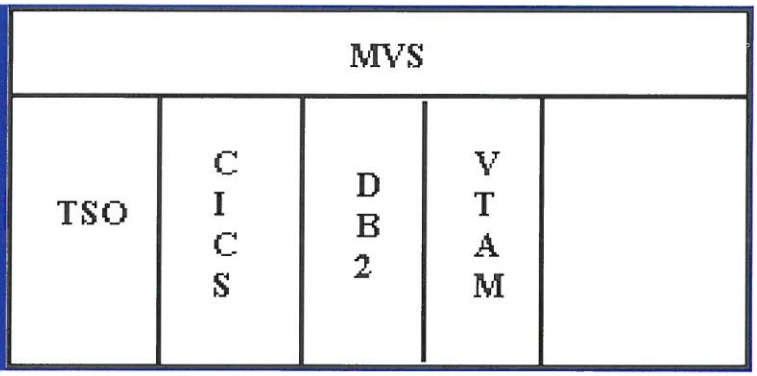


Figura 3: Regiones o espacios de direcciones del HOST.

los procesos *batch* fuera del horario de oficinas para que no compitan en el consumo de recursos de ordenador con los procesos ON-LINE en los que se busca procesar un único detalle.

Esto quiere decir que los procesos *on-line* permiten procesar programas muy cortos que tratan un único registro y que por tanto generan una contestación casi inmediata hacia el terminal que ha solicitado su ejecución. Un claro ejemplo de este tipo de procesos puede ser la reserva de un vuelo, o la disposición de dinero en un cajero automático.

En una primera aproximación, se podrían clasificar los distintos tipos de procesos *on-line* en los siguientes:

- Sistemas Interactivos para desarrollo y prueba de programas, tales como el TSO, ICCF.
- Sistemas DB/DC (*Data Base/Data Communication*). Estos sistemas permiten, con herramientas propias, construir programas y aplicaciones, y a su vez, proporcionan facilidades para el control de estas aplicaciones. Entre estos sistemas se encuentra el CICS y el IMS.
- Sistemas de Control. Son aquellos procesos que tienen por finalidad controlar procesos de fabricación específicos.

ENTORNO DEL CICS:

La figura 4 muestra el entorno de un sistema CICS. En ella se puede apreciar cómo este subsistema cuenta con:

- Servicios de control propios, que le permiten estar unido al sistema operativo para la utilización de los recursos que no son propiamente suyos.
- Funciones de manejo de datos, lo cual le permite aceptar, desde los programas de aplicación, peticiones de escritura o lectura sobre cualquier tipo de fichero, preparando las peticiones para que sean procesadas por el método de acceso apropiado.
- Funciones de comunicación, que le permiten realizar el control de terminales. Esta parte del CICS trata el sondeo y direccionamiento de todos los terminales y la transferencia de datos 'a' y 'desde' ellos.
- Servicios de programas, de aplicación, los cuales le permiten gobernar los programas asociados a cada aplicación dentro de la configuración propia del CICS.

CARACTERÍSTICAS PRINCIPALES

De entre las muchas características que se podrían enunciar, se pueden resaltar en una primera aproximación las siguientes:

- De cara al MVS representa un solo espacio de direcciones (*Address space*), tal y como se desprende de la figura 3.
- Se ejecutan todos los programas a la vez, solapándose las interrupciones generadas por las operaciones de E/S de una tarea con el proceso de otra petición, tal y como se puede deducir del estudio de la figura 2. Pero es el propio CICS quien asigna ciclos de CPU a otra tarea cuando se solicita una lectura por parte de la transacción que se está ejecutando.
- Así mismo, dispone de su propio *despachador* de tareas, asignando prioridades a cada una. No utiliza la gestión del *despachador* del MVS en sus tareas, ya que desde el punto de vista del MVS, el propio CICS es una única tarea.
- Realmente se ejecuta un solo programa *DFHSIP*, y todos los programas de usuario y del sistema son considerados subprogramas respecto a él.
- Los programas de usuario no están *linkeditados* con el programa principal, ya que el CICS tiene su propio cargador de programas.
- Desde un programa de aplicación, se tiene acceso a cualquier área de otro programa o de módulos básicos de CICS, lo que da lugar a la posibilidad de corrupción de áreas (violaciones de memoria).

CONCEPTOS IMPORTANTES

Transacción: Básicamente cada una de las peticiones que se recibe desde un terminal, constituye una transacción, la cual crea una unidad de trabajo o *TAREA* para la CPU. Esquemáticamente se podría decir que una transacción consiste en '*hacer una petición al host*'; éste ejecuta la petición y devuelve los resultados al terminal.

Tarea: Es la unidad de proceso interno para realizar una transacción. Siempre estará en uno de los dos estados siguientes:

- ACTIVA: se está ejecutando.
- SUSPENDIDA: Está esperando algo del sistema, por ejemplo que el sistema lea un registro de un fichero.

Multitarea: Múltiples aplicaciones se ejecutan al mismo tiempo en la misma región o espacio de memoria (ver figura 3).

Reentrabilidad, un programa se dice que es:

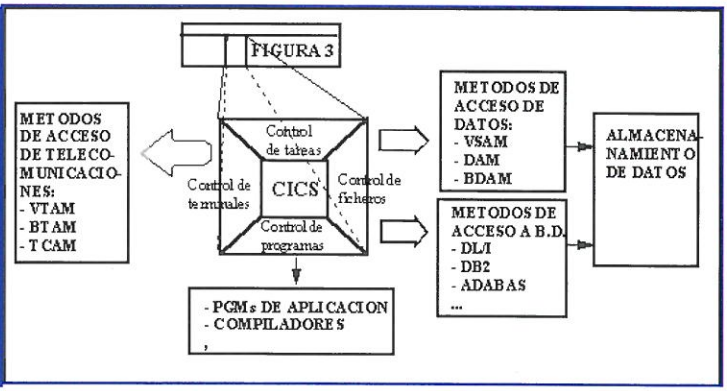


Figura 4: Entorno general del CICS.

■ **Reentrante**: cuando sólo hay un único módulo cargado en memoria, de modo que cuando una tarea termina de ejecutarlo, puede ser usado el mismo módulo por otra. Como es lógico suponer, los módulos reentrantes tienen una única *working* .

■ **Cuasi-reentrante**: Este tipo de programa *es 'serialmente reusable entre puntos de entrada y salida'*. Es decir, CICS asigna *workings* distintas a cada tarea que quiera usar dicho módulo concurrentemente, y así utiliza un único ejecutable. Ahora bien, sólo es reusable en serie (por varios) entre puntos de E/S, tal y como muestra la figura 5.

CICS es cuasi-reentrante, y esto no tiene nada que ver con el concepto conversacional que se trata a continuación.

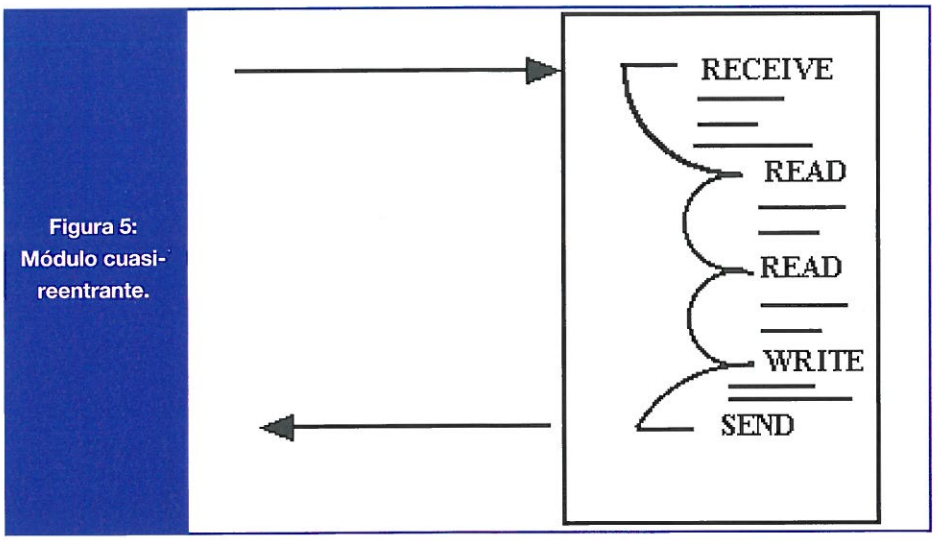


Figura 5: Módulo cuasi-reentrante.

Conversacionalidad:

Un programa se dice que es **conversacional** cuando cuenta con instrucciones para aceptar y mostrar datos, realizándose estas operaciones de entrada/salida tantas veces como lo necesite su ejecución para poder llegar al final y presentar los resultados solicitados (ver figura 6).

Este tipo de programas, como es fácil imaginar, inmoviliza áreas de memoria en las que se van almacenando los resultados intermedios.

Un programa se dice que es **pseudo-conversacional** cuando consta de una única instrucción para aceptar datos y una única instrucción para presentar datos, cediendo el control al CICS, de forma que la siguiente entrada de operador arranque una nueva tarea que ejecute otra transacción. Este tipo de programas inmoviliza menos áreas de memoria que el tipo conversacional y, para su ejecución total, requiere tantas tareas CICS como entradas haya que considerar. La *ventaja* es que mientras el operador 'se piensa la respuesta', en el ordenador central no existe tarea, pudiendo dedicarse esos recursos a otras tareas.

NATURAL

En artículos anteriores, en esta sección se ha tratado entre otros temas de NATURAL. El hecho de mencionar este producto ahora, es sólo para proporcionar un claro ejemplo de lo que realmente es CICS. En aquellos artículos se decía que NATURAL era un entorno, es decir un *MACROPROGRAMA* que permitía editar, crear comandos de usuario, programas, etc. Pues bien, en el caso de que NATURAL corra bajo CICS, NATURAL no es mas que una transacción CICS.

Así, NATURAL, como tal transacción CICS, se compone de una serie de rutinas que son invocadas por los servicios de control de programas del CICS, y que sirven para cargar y ejecutar los distintos ejecutables de los comandos NATURAL creados por el programador, exactamente igual que la transacción que solicita el saldo desde un cajero, y que se encuentra escrita en COBOL.

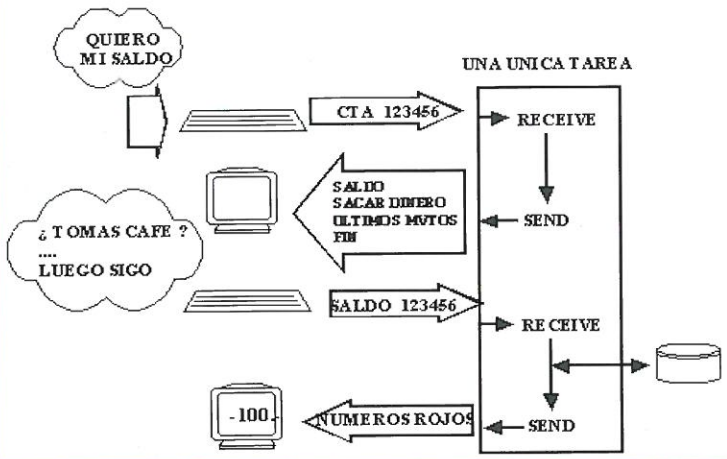


Figura 6: Transacción conversacional.

HISTORIA

Y ya que este articulo comenzó con algo de historia, se va a cerrar también con una pequeña comparativa de su evolución:

■ Tiene algo más de 20 años.

■ Fue diseñado en sus orígenes por una compañía eléctrica, (aunque el autor no ha podido confirmar este extremo), si bien IBM tuteló este proyecto casi desde sus comienzos.

■ Al principio era configurable, es decir cargaba o seleccionaba sólo los módulos con los que podía trabajar.

■ Contaba con una tabla de recursos.

■ Era excesivamente débil, (se caía muy fácilmente).

■ La recuperación era deficiente.

En la actualidad:

■ Es monobloque.

■ Cuenta con un fichero de recursos, el cual sustituye a las tablas que se verán en el próximo articulo.

■ Cuenta con un interface de programación en comandos.

■ Todavía es posible la corrupción de áreas comentada anteriormente.

■ La recuperación esta ya muy perfeccionada.

■ Es un producto estratégico de IBM para Grandes Sistemas.

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

Ramiro Carballo

Hasta ahora, en la programación tradicional, las máquinas sólo llevaban a cabo la aplicación de un algoritmo de cálculo o de gestión como contribución propia al proceso de solución de un problema. Con la Inteligencia Artificial, la Informática puede hacer frente a nuevos problemas gracias a las facilidades que aportan los nuevos planteamientos y técnicas de resolución.

El procesamiento del lenguaje natural, la enseñanza asistida por ordenador, las herramientas de programación automática, los sistemas expertos en general, el reconocimiento de patrones (voz, caracteres, olores, rostros), la planificación de tareas programando robots, los sistemas de producción en general, la teoría de juegos, etc., avanzan, día a día, en las universidades y centros de investigación de todo el mundo, ofreciendo resultados hasta ahora inimaginables.

EL LENGUAJE NATURAL

La posibilidad de "hablar con el ordenador", el gran sueño de muchos usuarios, está cada vez más próxima a convertirse en realidad, permitiendo que la máquina nos obedezca en nuestra forma normal de expresarnos y nos responda de igual manera.

Se pudieron contemplar, en el pasado SIMO-TCI en Madrid, los avances en interfaces de usuario que permiten la navegación por voz en un entorno operativo determinado con una fiabilidad del 99%.

Sin embargo, no debemos confundir esta tecnología con el reconocimiento del lenguaje natural, ya que en este caso se trata de reconocimiento de voz, área también incluida en los dominios de la Inteligencia Artificial.

Por hacer un símil con un compilador de un lenguaje de programación habitual, el reconocimiento de voz equivaldría al analizador léxico, capaz de extraer las características más elementales de la entrada y reconocer los *tokens* o palabras completas.

Tal vez, para un sistema de navegación por voz, no sea necesario más que esto, siendo el programador del interfaz el que lance uno u otro comando como respuesta al evento del reconocimiento de un *token* concreto.

Pero el tratamiento del lenguaje natural va más allá: consistiría en reconocer (o generar) frases completas de un idioma, pudiendo reconocer su sintaxis, y, tras el análisis de los componentes (sujeto, verbo, complementos, proposiciones, etc.), llegar a entender verdaderamente toda la semántica de la frase. Volviendo al ejemplo del compilador, se trataría del analizador sintáctico que recibe los *tokens* de un reconocedor de voz, unido a la acción del analizador semántico.

Las aplicaciones de una tecnología así no residen sólo en las interfaces de usuario, sino en aspectos mucho más cercanos. Por ejemplo, que los procesadores de texto no sólo realicen la corrección ortográfica sino también la sintáctica, después de la edición del usuario. También resultaría interesante poder transcribir conferencias o conversaciones de manera automática, tal y como vemos funcionar hoy los reconocedores de caracteres ópticos. Tampoco debemos olvidar un terreno en el que se están llevando a cabo grandes esfuerzos en investigación: los intérpretes (traductores de idiomas) simultáneos, que supondrán una revolución inmediata en el mercado.

Lo cierto es que se están alcanzando grandes metas en este área, pero todavía existe una frontera que superar: la mayoría de las soluciones se basan en la gramática de un determinado lenguaje (igual que en un compilador de un lenguaje de programación), que, naturalmente, tiene una extremada complejidad. El problema, una vez superada esta complejidad con las capacidades de proceso de las máquinas modernas, reside en que no existe todavía nin-

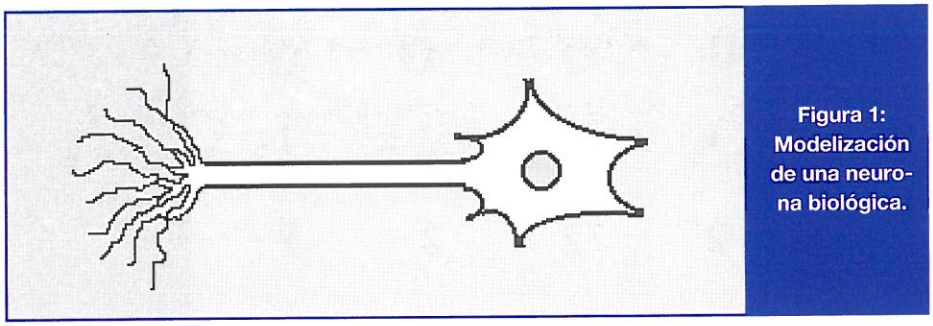


Figura 1: Modelización de una neurona biológica.

guna gramática completa de ningún idioma, debido principalmente al carácter subconsciente del lenguaje.

SISTEMAS EXPERTOS

La Inteligencia Artificial en general ha hecho grandes aportaciones a la informática, con las que ahora estamos familiarizados: las ventanas, los ratones, los iconos, por ejemplo, se usaron por primera vez en una máquina *Lisp*. La idea de interrumpir un programa durante su ejecución fue necesaria, originalmente, para suministrar la información de aprendizaje a la aplicación; además, existe la polémica entre los ingenieros del software y los del conocimiento sobre quién inventó las clases de la programación orientada a objetos, que en Inteligencia Artificial también se denominan marcos.

Pero si buscamos aportaciones de la Inteligencia Artificial a la sociedad, deberíamos destacar las aplicaciones de sistemas expertos.

Un sistema experto, es, simplemente, una herramienta a la que hemos sido capaz de transmitir, adecuadamente, los conocimientos "extraídos" a un experto humano.

El sistema experto no tiene porqué ser el núcleo de una aplicación en sí mismo; puede tratarse de una parte usada dentro de un programa algorítmico tradicional, para solucionar un aspecto concreto de difícil programación.

El típico ejemplo de sistema experto es aquel cuyo objetivo es acumular la experiencia de un médico. Cuando un paciente se presenta al sistema, le enumera los síntomas que padece. El experto (la máquina) busca en su base de datos de experiencias e intenta ajustar el cuadro presentado por el paciente con los síntomas de alguna enfermedad que conozca.

Si el ajuste no se produce totalmente, el sistema puede dudar entre varios diagnósticos. Este es el momento en que el ordenador puede preguntar al paciente sobre si padece algún síntoma concreto que le va a ayudar a decidir sobre uno u otro diagnóstico definitivo. Si, finalmente, el sistema no es capaz de diferenciar un diagnóstico con toda seguridad, podría ofrecer sus resultados con distintos porcentajes de probabilidad.

También cabe la posibilidad de que el sistema experto en medicina almacene los tratamientos aplicados a cada conjunto de síntomas y el porcentaje de éxito en los pacientes así tratados. En posteriores consultas, estas experiencias aportarán soluciones contrastadas, exactamente como lo haría un médico humano.

Otra aplicación habitual de los sistemas expertos es la realización de sistemas de ayuda a la decisión. *Hamlet*, por ejemplo, es un orientador vocacional para estudiantes preuniversitarios, que aporta el asesoramiento propio de un experto.

Podemos implementar, con un sistema experto, cualquier aplicación en la que resulte más sencillo acumular experiencia que programar, aunque hay bastantes ejemplos de sistemas que combinan ambas técnicas.

Por ejemplo, imaginemos el típico juego de ajedrez. Este juego puede ser programado estudiando la conveniencia o no de cada movimiento respecto a la probabilidad que me ofrezca de vencer al contrario (como se muestra más adelante).

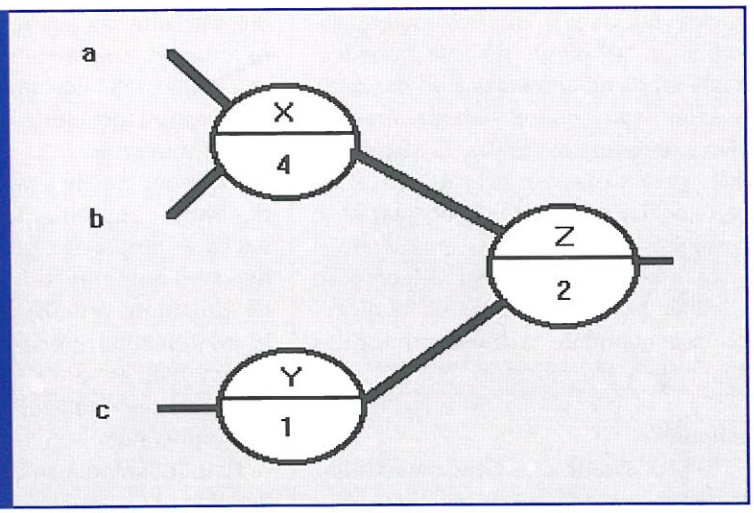
Sin embargo, en muchas ocasiones, se advierte que al programar no podemos contemplar varios casos, situaciones que no son fácilmente documentables y que denominamos *heurísticas* (por ejemplo, que en las tres en raya es mejor situar la ficha en la casilla central si está vacía).

Precisamente, estas *heurísticas* pueden ser aprendidas por un sistema experto integrado en el programa que controle el juego, al igual que lo hiciese un jugador humano de su contrincante.

Se debe destacar que un sistema experto, como cualquier sistema con aprendizaje, debe ser entrenado, y que los resultados obtenidos dependerán de la calidad de ese entrenamiento.

El lector se podrá ahora explicar los resultados de los recientes enfrentamientos entre un campeón mundial de ajedrez y un superordenador de la compañía de equipos líder en el mundo. Fue noticia que el jugador humano venció a la máquina, y así sirvió de mofa contra los informáticos. El problema no estaba en la capacidad del superordenador, sino en que los conocimientos, depositados por los expertos humanos en el sistema

Figura 2:



experto, eran inferiores a los del campeón mundial de ajedrez. Esperemos que el sistema haya aprendido para el próximo encuentro.

RECONOCIMIENTO DE PATRONES

Se engloba en este área el reconocimiento de cualquier patrón, caracteres ópticos, olores, objetos en el espacio, rostros, voz, etc.

La diferencia entre uno u otro proceso de reconocimiento se centra, principalmente, en los sensores utilizados para percibir el medio y en las técnicas empleadas para extraer las características del elemento a reconocer. Si se trata de olores, se utilizarán unos sensores químicos. Si son caracteres de texto, bastará con un escáner. Para rostros y objetos en el espacio, necesitaremos una cámara, y para voz, un micrófono.

Sea cual sea el sensor utilizado, el siguiente paso será la digitalización de las señales continuas recibidas. En el caso del escáner, la digitalización se produce en el propio dispositivo, pero si las cámaras de vídeo son analógicas (porque son más rápidas que las digitales) se debe convertir en una señal discreta para poder ser procesada por el ordenador.

En este momento, lo único que se conoce del elemento a reconocer es un conjunto de números (resultado de la digitalización) que se agrupará en un vector llamado de características discriminantes.

La dificultad del reconocimiento de patrones reside en la etapa de clasificación. Supongamos un sistema de reconocimiento de voz, que sólo quiere distinguir un "sí" de un "no". Se procederá a una etapa de aprendizaje en la que un usuario jugará con el sistema, introduciéndole muestras de voz. El sistema le dirá, para cada muestra, lo que él ha reconocido, y el usuario le aceptará o le corregirá, dado que nos encontramos en la fase de entrenamiento. El proceso continúa hasta que el sistema ha definido correctamente la frontera entre las dos clases de muestras (el "sí" y el "no") y el porcentaje de error resulte aceptable.

El proceso de clasificación se realiza estableciendo fronteras entre clases, es decir, entre conceptos que queremos reconocer, pero con la dificultad de que muchas muestras, distintas, de la realidad pertenecen a una misma clase, y lo normal es que las fronteras entre clases sean bastante difusas para el clasificador.

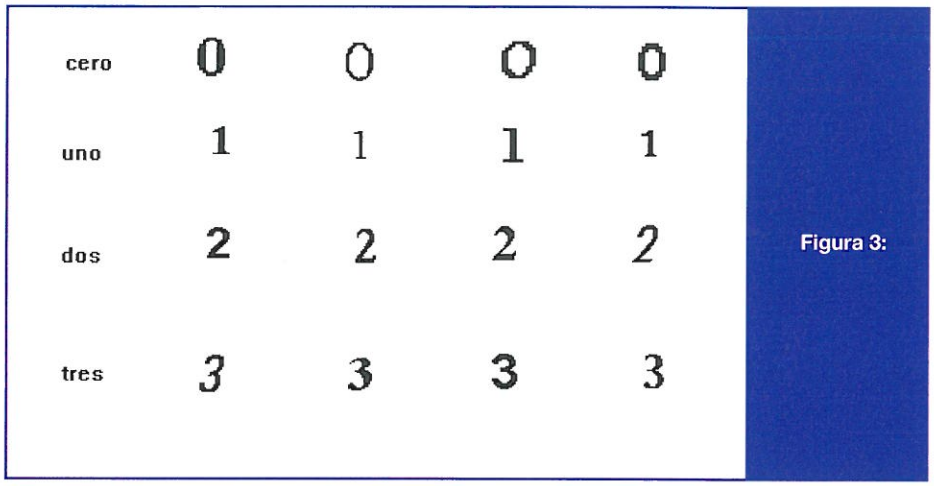


Figura 3:

Por ejemplo, el sistema que antes se había entrenado para distinguir un "sí" de un "no" podría fallar (por encima de su porcentaje de error aceptable) si el usuario sufre un resfriado, ya que las características de las muestras han variado respecto de aquellas con las que aprendió el sistema. No hablemos ya del caso en el que fuese otro usuario el que probase el sistema.

En el caso de un OCR, deberíamos entrenar a nuestro sistema para que distinguiese tantas clases como caracteres se deban reconocer. Se podría comprobar que si se reconoce bien el carácter "a", le puede costar más reconocer "*a*" o simplemente la misma letra con otra fuente. Otro caso habitual en el que no se delimitan suficientemente las fronteras entre clases es cuando nuestro sistema confunde el carácter "ó" con una "d", debido a la similitud de ciertas características de las muestras.

Para hacer clasificaciones se emplean distintas implementaciones. Unas veces se emplean algoritmos de clasificación automática (que trataremos en próximos artículos), pero cuando la complejidad crece (en número de características o de clases a distinguir) se suelen utilizar redes de neuronas artificiales.

Esta tecnología se basa en la reproducción del funcionamiento de las neuronas biológicas, intentando imitar la sinopsis con que se conectan mediante la asignación de umbrales de excitación a las unidades que componen la red neuronal, y de pesos a las conexiones que se establecen entre ellas y que valoran su importancia.

Las redes de neuronas se utilizan en sistemas de reconocimiento de caracteres, y también , con bastante éxito, de rostros. En este último, se le proporcionan al sistema diez imágenes distintas de la cara de una persona, durante la etapa de entrenamiento. La complejidad reside en la extracción de características que puedan ser más discriminantes de los rostros.

Es evidente que hay que enseñar al sistema cada vez que se quiera que reconozca una nueva persona. En este caso existirán tantas clases distintas como rostros se deseen reconocer.

Las técnicas de visión artificial implementadas en robots se engloban también en este área.

TEORÍA DE JUEGOS

Los juegos de ajedrez pueden ser un buen ejemplo. Se empieza por modelizar los movimientos de las piezas. En aplicaciones comerciales se encuentran dos tipos de implementaciones: basadas en la programación de cada tipo de pieza por separado, contemplando los casos de casillas libres u ocupadas por el contrincante o por piezas del propio jugador, así como los límites del tablero; la segunda solución y más cercana a la

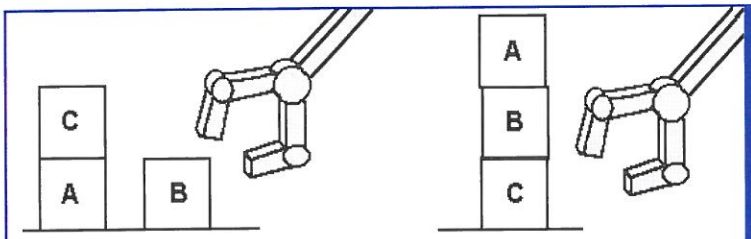


Figura 4:

Inteligencia Artificial es la utilización de sistemas de producción.

Con este sistema el programador no se tiene que preocupar de la contemplación de tantos casos (con tantos *if*'s anidados), sino de generar una base de hechos en la que almacene la configuración de las posibles situaciones que se pueden dar en el juego. Existirá un motor de interferencias capaz de encontrar cuál de esas configuraciones se ajusta en cada momento a la situación actual de la partida y podrá actuar en consecuencia, disparando una regla asociada.

Lo habitual es que se puedan disparar varias reglas a la vez, lo que acaba produciendo una explosión combinatoria que hay que controlar. Para ello, el sistema de producción utiliza estrategias de control para decidir cuál de las posibles reglas lanzará. Ese control puede venir dado, en ocasiones, por funciones de evaluación.

Estas funciones evalúan la situación en la que se encuentra ahora la partida y, a continuación, se simula la ejecución de todas las reglas posibles que se hubiesen podido disparar y se evalúa la situación a la que nos llevaría cada una de ellas (que implican el movimiento de una pieza en el tablero).

Aplicando algoritmos de búsqueda, se determina cuál de las posibles jugadas me proporcionaba un valor de la función de evaluación más óptimo, es decir, con mayor probabilidad de conseguir la victoria.

PLANIFICACION DE TAREAS

Si antes se trató la visión artificial de los robots, en este área se centra ahora la programación de esos robots de manera dinámica respecto del entorno percibido. Además, la planificación simbólica de tareas se utiliza en la vida real en aplicaciones que van desde la planificación de procesos hasta el establecimiento de rutas en las empresas de logística.

Se define como el conjunto de procesos que se realizan a fin de obtener un plan de actuación para poder resolver un problema. Lo normal es que la salida de un proceso de planificación pueda introducirse en un robot de forma que ejecute el plan obtenido. Para ello debemos proporcionarle al planificador un estado inicial, un conjunto de funciones de transición entre estados y un estado final o meta. Entonces el planificador o proceso de planificación realizará una búsqueda en un espacio de estados y nos devolverá la solución en forma de plan.

Para ilustrar las posibilidades de un planificador vamos a estudiar un problema descrito en muchos libros de Inteligencia Artificial. [Nilsson, 1980, Rich and Knight, 1991 y D. Borrajo, 1993]. Se trata del mundo de los bloques, el más utilizado, ya que, a pesar de ser conceptualmente sencillo, plantea problemas muy difíciles a los sistemas de planificación.

En todo planificador, primero se debe especificar el dominio, indicando un conjunto de estados válidos y unos operadores (funciones de transición). El mundo de los bloques consiste en un conjunto de bloques encima de una mesa, que pueden estar o no unos sobre otros. También hay un brazo de robot que puede coger uno y sólo uno de los bloques al tiempo.

El brazo de robot puede hacer las siguientes operaciones: levantar un objeto de la mesa, dejar un objeto en la mesa, quitar un objeto de encima de otro y poner un objeto sobre otro.

Después de especificar el dominio sobre el que se va a trabajar, se le plantea un problema concreto al planificador: un estado inicial y una meta a conseguir. Planteemos el problema conocido como anomalía de Sussman en el que inicialmente se encuentran sobre la mesa los bloques A y B, y C está encima de B, y la meta es que los tres formen una torre, con A arriba, C abajo y B en medio.

La solución obtenida por el planificador nos indica los pasos que debemos seguir:

1. Quitar C de encima de A.
2. Dejar C sobre la mesa.
3. Coger B de la mesa.
4. Poner B sobre C.
5. Coger A de la mesa.
6. Poner A sobre B.

El ejemplo del mundo de los bloques es un caso muy concreto destinado al estudio de la planificación, pero no debemos descartar la utilidad de este área de la Inteligencia Artificial en todo aquello que necesite instrucciones para alcanzar una solución: la relación de estaciones y transbordos que debemos seguir en el metro para llegar a un punto de la ciudad si partimos de otro o incluso un demostrador automático de teoremas son otras aplicaciones útiles de la planificación.

CONCLUSIÓN

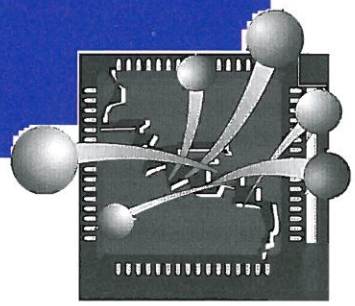
Las aplicaciones de la Inteligencia Artificial se extienden cada vez más y en las áreas más insospechadas.

En el próximo número, se profundizará en los fundamentos de los sistemas de producción (base de la mayoría de implementaciones sobre Inteligencia Artificial). Estos sistemas nos servirán de conexión con la teoría de juegos, descubriendo al lector nuevas formas de programación y nuevos conceptos.

Contactar con al autor:
carballo@lander.es

LOS OBJETOS

Juan Manuel y Luis Martín



El método utilizado por la mayoría de las aplicaciones Windows profesionales para organizar y facilitar el acceso a la selección de opciones es la incorporación de sistemas de menús. En este artículo se estudiará tanto la forma de crear un sistema de menús en tiempo de diseño, como la forma de alterar y controlar su funcionamiento en tiempo de ejecución.

A medida que las aplicaciones son más complejas, su manejo va requiriendo un mayor número de decisiones por parte del usuario. Cuando el número de opciones entre las que debe seleccionar el usuario es muy elevado, la utilización de controles tales como casillas de verificación, botones de opción y botones de comando resulta inadecuada. Por tanto, se hace necesario dotar a las aplicaciones de un sistema que permita organizar adecuadamente las distintas opciones y facilitar su acceso al usuario.

La mejor forma de organizar las opciones de usuario consiste en dotar a la aplicación de un sistema de menús. Como el lector ya sabe, los sistemas de menús consisten en conjuntos organizados de listas desplegables que presentan al usuario diversas opciones, cada una de ellas capaz de desencadenar una determinada acción. El acceso a dichas opciones puede realizarse tanto con el ratón como con el teclado, de una forma sencilla e intuitiva.

Para asociar a un formulario un sistema de menús, Visual Basic dispone de un tipo especial de controles, denominados *controles de menú*. Cada uno de los elementos que constituyen el sistema de menús, ya sean éstos títulos u opciones, son controles de menú y, como cualquier otro control, disponen de su propio repertorio de propiedades. Sin embargo, los controles de menú no disponen de ningún método, y sólo pueden responder al evento *Click*.

Cuando se asocian controles de menú a un formulario, debajo de la barra de títulos del mismo aparece una nueva barra, denominada *Barra de menús*. Sobre ella se irán situando los distintos menús de que constará la aplicación.

EL EDITOR DE MENÚS

Para crear un sistema de menús asociado al formulario actual en tiempo de diseño, es necesario acceder al *Editor de menús*. Se trata de una ventana que permite añadir controles de menú al formulario, así como especificar la situación jerárquica y las propiedades de cada uno de ellos. El acceso puede realizarse seleccionando la opción *Editor de menús* en el menú *Herramientas*, o pulsando *[Ctrl+E]*.

La *figura 1* muestra el aspecto del editor de menús. Como puede verse, consta de dos partes claramente diferenciadas. En la parte inferior aparece la *Lista de controles*, que contiene todos los controles de menú asociados al formulario actual. En la parte superior aparece un grupo de controles que permiten especificar las características de cada uno de los elementos del sistema de menús, es decir, las propiedades de los controles de menú.

Como ya se ha mencionado, cada elemento del sistema de menús, ya sea un título o una opción, constituye un control de menú. La lista de controles permite seleccionar el control cuyas características se desean especificar. Para seleccionar un elemento de la lista, bastará con hacer clic sobre él, apareciendo entonces resaltado, y mostrándose sus propiedades en los controles de la parte superior.

El texto que aparece representando a cada uno de los controles dentro de la lista, se corresponde con el texto que será visualizado en tiempo de ejecución. Así mismo, la posición que ocupa cada elemento dentro de la lista determina el tipo de elemento:

- Los elementos de la lista que aparecen alineados en el borde izquierdo representan los títulos de cada menú.
- Los elementos de la lista que aparecen debajo de cada título, indentados hacia la derecha, son las opciones de ese menú.

- Los elementos de la lista que aparecen simplemente como un guión (-) son las *barras separadoras*.

Para facilitar el manejo de la lista de controles, existe un conjunto de botones que aparecen inmediatamente sobre ella:

- Flechas *Arriba* y *Abajo*: Permiten cambiar la posición del elemento actualmente seleccionado dentro de la lista.
- Flechas *Izquierda* y *Derecha*: Permiten cambiar el nivel de indentación del elemento actual.
- *Siguiente*: Selecciona el siguiente elemento de la lista.
- *Insertar*: Inserta un elemento nuevo entre el actual y el anterior.
- *Eliminar*: Elimina el elemento actualmente seleccionado.

Los controles que aparecen en la sección propiedades del control permiten especificar las características del elemento actualmente seleccionado, es decir, las propiedades del control de menú. La *tabla 1* muestra el significado de cada uno de esos controles, y la propiedad que les corresponde.

Al igual que sucede con la mayoría de los controles, la propiedad *Caption* indica el texto que será visualizado en tiempo de ejecución representando al control. En esta propiedad suele incluirse un carácter *ampersand* (&) para definir una tecla de acceso, tal y como se vio para el resto de controles. También es habitual en las aplicaciones Windows que el nombre de una opción de menú acabe en puntos suspensivos (...) cuando la acción desencadenada consiste en desplegar un cuadro de diálogo.

Así, por ejemplo, para crear un sistema de menús que contenga un menú *Archivo* con las opciones *Abrir*, *Cerrar* y *Salir*, y otro menú *Ayuda* con las opciones *Contenido* y *Acerca de*, deben crearse 7 controles de menú y asignarles los valores adecuados a su propiedad *Caption*. Estos valores serán mostrados en la lista de controles del editor de menús, que pasará a tener el siguiente aspecto:

&Archivo
  &Abrir
  &Cerrar
  &Salir
Ay&uda
  &Contenido
  &Acerca de

Como ocurría con el resto de controles, el nombre que representa al control de menú en el código se especifica en la propiedad *Name*. Aquí también es recomendable la utilización de un prefijo identificativo del tipo de control, que generalmente suele ser *mnu*.

Una vez creados los controles de menú con el editor de menús, la especificación de sus propiedades también puede realizarse a través de la ventana de propiedades. Dentro de ella, los controles de menú del formulario aparecen como cualquier otro control.

ACCESO POR TECLADO

Además de utilizando el ratón, el acceso a cada una de las opciones de los diferentes menús puede realizarse también mediante el uso de unas teclas predefinidas en el proceso de diseño del sistema de menús. Se trata de las *teclas de acceso* y las *teclas rápidas*.

El funcionamiento de las teclas de acceso es idéntico al explicado en artículos anteriores para el resto de controles. Para ello, bastará con incluir el carácter *ampersand* (&) en la propiedad *Caption*, delante de la letra correspondiente a la tecla deseada.

Para abreviar aún mas el proceso de ejecución de las opciones de un menú, es posible asignar una tecla rápida (*Shortcut Key*) a cada una de dichas opciones. De esta forma, cuando la tecla es pulsada, la acción se desencadena inmediatamente, sin necesidad de desplegar el menú correspondiente. La tecla rápida puede ser un tecla individual o una combinación de teclas, como *[Ctrl+F1]*, *[Shift+F2]*, etc. En tiempo de ejecución, la descripción de la tecla rápida que corresponde a una opción del menú se visualiza a la derecha del texto de la opción.

La asignación de una tecla rápida a una opción de menú se realiza en el editor de menús, seleccionando el control correspondiente en la lista de controles. Seguidamente, se despliega la lista *Shortcut* y se selecciona una tecla o combinación de teclas. La *figura 2* muestra desplegado el menú *Editar* del bloc de notas. Como puede verse, las opciones *Deshacer*, *Cortar*, *Copiar*, *Pegar*, *Eliminar* y *Hora/Fecha* tienen asignadas las teclas rápidas *[Ctrl+Z]*, *[Ctrl+X]*, *[Ctrl+C]*, *[Ctrl+V]*, *[Supr]* y *[F5]*, respectivamente.

SUBMENÚS

Cuando un control de menú que representa un título no tiene opciones asociadas, es capaz de desencadenar directamente una acción. Igualmente ocurre con las distintas opciones de cada menú. Sin embargo, también es

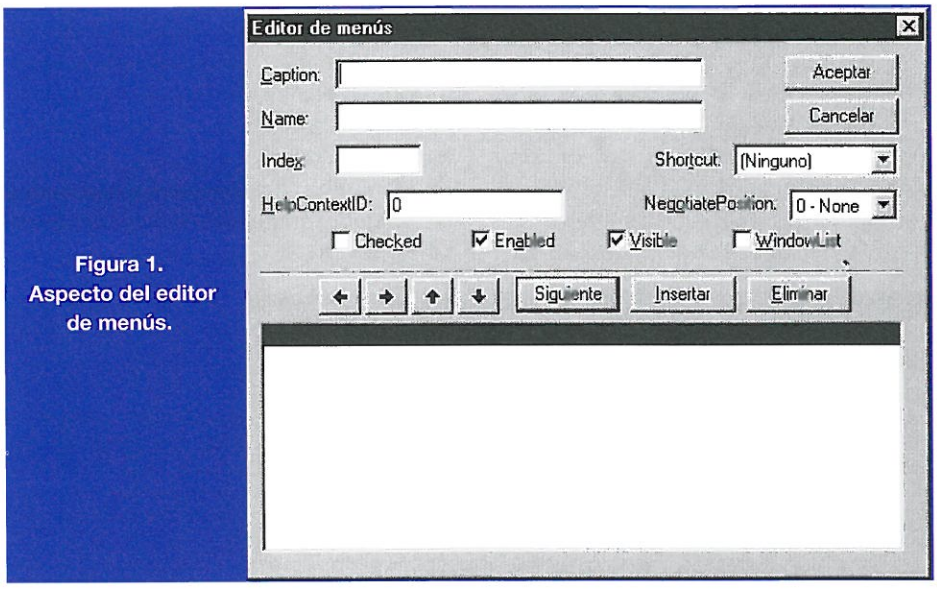


Figura 1. Aspecto del editor de menús.

TABLA 1

Propiedad	Descripción
Caption	Literal que será visualizado para ese elemento en el sistema de menús.
Name	Nombre que será usado como referencia del elemento dentro del código, es decir,el nombre que determinará el procedimiento de evento asociado.
Index	Número asignado al control de menú cuando éste forma parte de una matriz de controles.
Shortcut	Tecla rápida para la ejecución de la opción correspondiente.
HelpContextID	Número que identifica la ayuda relativa al elemento.
NegotiatePosition	Especifica cómo será visualizado el elemento dentro de la barra de menús.
Checked	Indica si debe visualizarse el símbolo *checkmark* (√) a la izquierda del elemento.
Enabled	Indica si el elemento está o no activo, es decir, si debe responder al procedimiento de evento correspondiente.
Visible	Especifica si el elemento debe o no visualizarse.
WindowList	Indica si el control debe contener una lista con los formularios hijos que se encuentran abiertos (sólo para formularios MDI)

Controles de la lista de propiedades del editor de menús.

posible hacer que una opción, en vez de desencadenar una acción, despliegue a su vez nuevas opciones.

Cuando las opciones de un menú despliegan a su vez nuevas opciones, en vez de realizar una determinada acción, constituyen lo que se denomina un *submenú*. De esta forma, cuando son seleccionadas, se despliega una nueva lista de opciones. Estas opciones de submenú, a su vez, pueden constituir nuevos submenús, y así sucesivamente. Visual Basic permite crear hasta cinco niveles de submenús.

Como siempre, para desplegar un submenú basta con situar el puntero del ratón sobre él. Como puede observarse, aquellas opciones que despliegan a su vez nuevas listas de opciones, es decir, que constituyen submenús, vienen marcadas con el símbolo u a su derecha.

Para la creación de submenús debe accederse al editor de menús. En la lista de controles, aquellos elementos indentados hacia la derecha que, a su vez, tienen debajo nuevos elementos indentados un nivel más, no son simples opciones de menú, sino submenús. Los elementos que aparecen debajo de ellos son las distintas opciones del submenú. Los distintos niveles de indentación indican el nivel del submenú. La figura 3 muestra cinco niveles de submenús desplegados.

Para realizar los niveles de submenús de dicha figura, será necesario crear una estructura en la lista de controles del editor de menús como la siguiente:

```
&Archivo
&Información
  &Publicidad
  &Noticias
    &Internacional
    &Economía
    &Deportes
      &Atletismo
      &Baloncesto
      &Futbol
        &Resultados
        &Clasificaciones
          &1ª División
          &2ª División
```

En fase de diseño, la estructura de un sistema de menús es fácilmente modificable. Así, es posible cambiar el título de las opciones, su nivel de indentación, sus teclas de acceso y rápidas, etc. Para ello, debe accederse al editor de menús y hacer uso de las distintas herramientas de que éste dispone.

SEPARADORES

Visual Basic permite separar las distintas opciones de un menú o submenú mediante *barras separadoras*. Estos separadores no tienen ningún efecto operativo, pero permiten agrupar las distintas opciones con el fin de facilitar su identificación.

Las barras separadoras son también elementos de la lista de controles. Por tanto, deberán insertarse entre los controles de menú de las opciones a separar. Para que un elemento de la lista de controles constituya una barra separadora debe especificarse un guión (-) en su propiedad *Caption*.

Al igual que los otros elementos de la lista de controles, las barras separadoras también deben tener un nombre distinto del resto, es decir, un valor único en su propiedad *Name*. Sin embargo, este valor es irrelevante, ya que no será utilizado en el código. Una convención bastante usual para nombrarlas consiste en denominarlas *mnuSep1*, *mnuSep2*, y así sucesivamente.

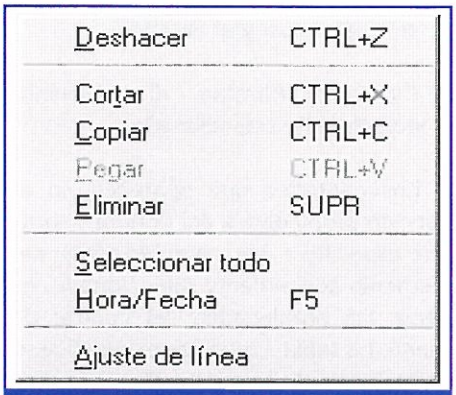


Figura 2.
Menú con teclas rápidas asociadas.

Así, por ejemplo, para separar las opciones *Guardar* y *Salir* de un menú con una barra, puede usarse la siguiente estructura:

TABLA 2

Valor	Constante	Descripción
0	*vbPopupMenuLeftAlign*	El argumento X especificará la coordenada de la esquina izquierda del menú emergente
4	*vbPopupMenuCenterAlign*	El argumento X especificará la coordenada del centro del menú emergente
8	*vbPopupMenuRightAlign*	El argumento X especificará la coordenada de la esquina derecha del menú emergente
0	*vbPopupMenuLeftButton*	Las opciones del menú emergente sólo pueden seleccionarse con el botón primario del ratón
2	*vbPopupMenuRightButton*	Las opciones del menú emergente pueden seleccionarse con ambos botones del ratón

Valores de los indicadores del método PopupMenu.

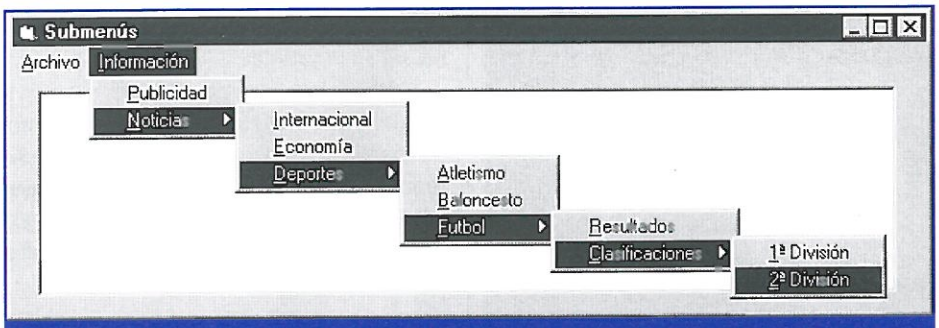


Figura 3. Distintos niveles de submenús desplegados.

&Guardar
-
&Salir

CONTROL DE MENÚS

Los menús de las aplicaciones realizadas con Visual Basic permiten realizar ciertas operaciones que modifican su aspecto y funcionamiento tanto en fase de diseño como en tiempo de ejecución. Permite adaptarlos a las necesidades del usuario en cada momento durante la ejecución de la aplicación. Las operaciones que pueden realizarse para modificar y controlar tanto el aspecto como el funcionamiento de un sistema de menús son las siguientes:

- Visualizar marcas sobre las opciones para indicar determinadas situaciones.
- Deshabilitar opciones del menú, de forma que el usuario no pueda ejecutarlas.
- Ocultar opciones del menú, de forma que no sean visibles para el usuario.

Tanto en fase de diseño como en tiempo de ejecución, es posible hacer que las opciones de un menú se visualicen marcadas, indicando determinadas condiciones relativas al estado de dicha opción. Cuando esto ocurre, se visualiza el símbolo *checkmark* (√) a la izquierda del texto de la opción. Este tipo de marcas permite reflejar estados internos de la aplicación del tipo Si/No, On/Off, etc.

Para visualizar una marca en un elemento del menú será necesario poner a *True* la propiedad *Checked* del mismo. Igualmente, para eliminar la marca será necesario poner a *False* dicha propiedad. En fase de diseño, el acceso a esta propiedad puede realizarse a través del editor de menús o de la ventana de propiedades. En tiempo de ejecución se realiza como para cualquier otro control:

```
mnuArchCerrar.Checked = True
'Marcar la opción Cerrar
mnuArchCerrar.Checked = False
'Desmarcarla
```

Al igual que otros controles, las opciones de un menú pueden ser deshabilitadas, de forma que, aún estando visibles, el usuario no pueda ejecutarlas. En tiempo de ejecución, el texto de las opciones que han sido deshabilitadas se visualiza difuminado. Así, por ejemplo, en la figura 2, la opción *Pegar* ha sido deshabilitada. Para *deshabilitar* una opción de menú será necesario poner a *False* su propiedad *Enabled*. Para volver a habilitarla será necesario poner a *True* dicha propiedad.

```
mnuArchCerrar.Enabled = False
'Deshabilitar la opción Cerrar
mnuArchCerrar.Enabled = True
'Volver a habilitarla
```

En ciertas ocasiones puede interesar que una opción del menú, además de no ser accesible por el usuario, tampoco se visualice. En este caso, debe ponerse a *False* la propiedad *Visible* del control de menú correspondiente. Para restaurar la opción bastará con volver a poner a *True* dicha propiedad.

CUADRO 1: EJEMPLO DE ADICIÓN Y ELIMINACIÓN DINÁMICA DE OPCIONES.

1. Añadir al formulario dos botones de comando y un cuadro de texto, tal y como se muestra en figura 4.

2. Añadir la siguiente estructura de menús:

```
&Título          'Name: mnuTitulo
&Salir      'Name: mnuSalir
-                'Name: mnuOpciones
```

3. Especificar en tiempo de diseño las siguientes propiedades:

Form1	Name	frmMenu
	Caption	Ejemplo de menú dinámico
Text1	Name	txtOpción
	Text	
Command1	Name	cmdAñadir
	Caption	&Añadir
Command2	Name	cmdEliminar
	Caption	&Eliminar
mnuOpciones	Index	0

4. Escribir las siguientes declaraciones generales y procedimientos de evento:

```
Dim Opciones As Integer

Private Sub mnuSalir_Click()
  Unload Me
End Sub

Private Sub mnuOpciones_Click(Index As Integer)
  MsgBox mnuOpciones(Index).Caption, , "Opción seleccionada"
End Sub

Private Sub cmdAñadir_Click()
  Opciones = Opciones + 1
  Load mnuOpciones(Opciones)
  mnuOpciones(Opciones).Caption = txtOpcion.Text
  mnuOpciones(Opciones).Visible = True
End Sub

Private Sub cmdEliminar_Click()
  If Opciones > 0 Then
    Unload mnuOpciones(Opciones)
  End If
End Sub
```

5. Salvar el formulario en el archivo MENU.FRM y el proyecto en el archivo MENU.VBP.

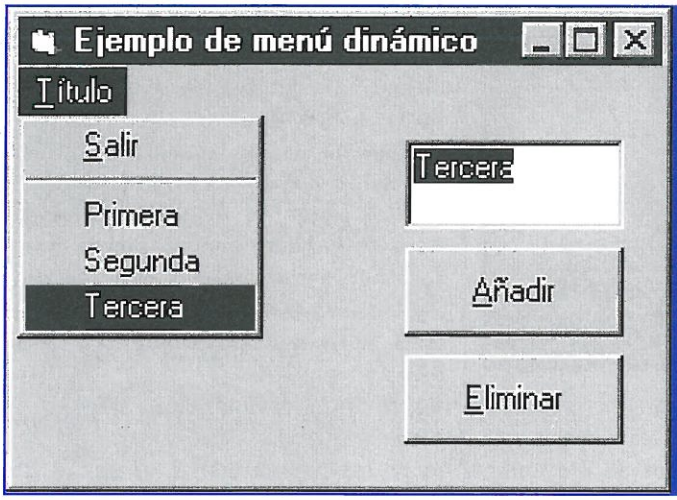


Figura 4. Ejemplo de menús dinámicos.

```
mnuArchCerrar.Visible = False
'Ocultar la opción Cerrar
mnuArchCerrar.Visible = True
'Volver a visualizarla
```

Cuando se oculta una opción de un menú, el resto de las opciones se desplaza una posición hacia arriba. Si lo que se oculta es el título de un menú, todas las opciones que dependen de él son ocultadas y, por tanto, inaccesibles para el usuario.

CÓDIGO PARA MENÚS

Al seleccionar una opción de un menú, bien sea haciendo clic sobre ella, pulsando su tecla de acceso o su tecla rápida, se produce el evento *Click* asociado al control de menú y, por tanto, se ejecutará el procedimiento de evento correspondiente. Para que la selección de la opción lleve a cabo las acciones deseadas, debe escribirse el código apropiado en el procedimiento de evento *Click* del control de menú que representa dicha opción.

Como ya se ha mencionado anteriormente, la propiedad *Name* de cada control de menú determina el nombre con que se identificará a dicho objeto a nivel de código. Por tanto, si la propiedad *Name* de un control de menú contiene el nombre *mnuArchivo*, el procedimiento de evento que debe escribirse será *mnuArchivo_Click*.

El acceso a los procedimientos de evento de los controles de menú puede realizarse a través de la ventana de código, de forma similar a como se hace para cualquier otro control. Sin embargo, en fase de diseño los menús que van siendo creados son visualizados sobre el formulario cuando se cierra el editor de menús. De esta forma, si se selecciona un determinado elemento del menú, se accede directamente a su procedimiento de evento *Click* dentro de la ventana de código.

MATRICES DE MENÚS

Con Visual Basic es posible hacer que varias de las opciones de un determinado menú compartan el mismo procedimiento de evento. A estos grupos de opciones se les denomina *matrices de menús* y su funcionamiento es muy similar al de las matrices de controles de otros tipos, tal y como se explicó en artículos anteriores.

La utilización de matrices de menús simplifica el desarrollo del código, ya que permite compartir fragmentos del mismo a los diversos controles de menú. Además, permite añadir nuevas opciones (y eliminarlas) en tiempo de ejecución.

En una matriz de controles de menú, cada uno de ellos tendrá asociado un número de orden en su propiedad *Index*. Este número es pasado como argumento al procedimiento de evento cuando la opción es ejecutada. De esta forma, el código del procedimiento es capaz de detectar qué opción es la que ha sido ejecutada.

Como siempre, para crear matrices de menús es necesario acceder al editor de menús. Los diversos elementos de la matriz se crean de la misma forma que ya ha sido estudiada. Sin embargo, para crear una matriz de menús debe tenerse en cuenta lo siguiente:

- Todos los elementos que pertenecen a la matriz deben estar situados en posiciones contiguas dentro de la lista de controles, e indentados al mismo nivel.
- El nombre especificado en la propiedad *Name* de todos los elementos de la matriz debe ser el mismo.
- Para cada uno de los elementos de la matriz debe especificarse un valor distinto de la propiedad *Index* (0 para el primero, 1 para el segundo, y así sucesivamente).

Cuando en la fase diseño se selecciona cualquiera de las opciones pertenecientes a una matriz de menús, el procedimiento de evento *Click* que se edita en la ventana de código es siempre el mismo. Sin embargo, éste contendrá la declaración del parámetro *Index* en su cabecera, permitiendo diferenciar la opción que ha sido ejecutada.

Cuando en tiempo de ejecución se seleccione una de las opciones pertenecientes a la matriz de menús, al procedimiento de evento común se le pasará como argumento el valor del índice correspondiente a la opción seleccionada, con el fin de poder identificarla. Por tanto, dentro del procedimiento será necesario utilizar sentencias condicionales para realizar el tratamiento apropiado de cada opción.

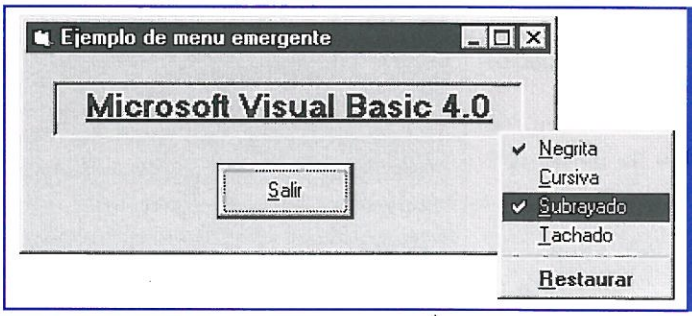


Figura 5. Ejemplo de menú emergente.

Así, por ejemplo, puede utilizarse una estructura *Select Case* para realizar las distintas acciones correspondientes a cada opción, en función del valor del argumento *Index*.

```
Private Sub
mnuArchMatriz_Click(Index As
Integer)
  Select Case Index
    Case 0
      'Acciones para abrir un archivo
    Case 1
      'Acciones para guardar el archivo
    Case 2
      'Acciones para cerrar el archivo
  End Select
End Sub
```

En tiempo de ejecución, Visual Basic permite añadir y eliminar opciones de menú dinámicamente. De esta forma, los menús pueden crecer o disminuir según las necesidades del usuario. La principal limitación de este proceso, es que los elementos añadidos o eliminados deben formar parte siempre de una matriz de menús.

Para añadir un elemento a una matriz de menús debe utilizarse la sentencia *Load*, ya que ésta permite cargar formularios y controles en la memoria. En el caso de los controles de menú, será necesario especificar el nombre de la matriz y el índice del nuevo elemento que se desea cargar. La sintaxis puede resumirse de la siguiente forma:

Load [Formulario.]MatrizMenús(Índice)

Una vez que el control ha sido cargado en la memoria, será necesario especificar sus distintas propiedades (*Caption*, *Enabled*, *Checked*, etc) con el fin de adaptarlo a la situación actual de la aplicación. Además, debe tenerse en cuenta que los controles añadidos mediante la sentencia *Load*, se cargan con la propiedad *Visible* a *False*. Por tanto, será necesario ponerla a *True* para visualizarlos.

Para eliminar un control del menú en tiempo de ejecución pueden utilizarse dos métodos. El primero consiste en poner a *False* su propiedad *Visible*. Sin embargo, este método simplemente se limita a ocultar la opción al usuario. Si se desea eliminar

CUADRO 2.- EJEMPLO DE UTILIZACIÓN DE MENÚS EMERGENTES.

1. Añadir al formulario un control etiqueta y un botón de comando, situándolos de forma similar a como se muestra en la figura 5.

2. Añadir al formulario la siguiente estructura de menús:

```
&Contexto                'Name: mnuContexto
  &Negrita               'Name: mnuNegrita
  &Cursiva               'Name: mnuCursiva
  &Subrayado     'Name: nmuSubrayado
  &Tachado       'Name: mnuTachado
  -              'Name: mnuSeparador
  mnuRestaurar   'Name: mnuOpciones
```

3. Especificar en tiempo de diseño las siguientes propiedades:

Form1	Name	frmContexto
	Caption	Ejemplo de menú emergente
Label1	Name	lblDemo
	Caption	Microsoft Visual Basic 4.0
	Alignment	2 - Center
	BorderStyle	1 - Fixed Single
	Font.Size	14
Command1	Name	cmdSalir
	Caption	&Salir
mnuContexto	Visible	False

4. Escribir los siguientes procedimientos de evento:

```
Private Sub cmdSalir_Click()
  Unload Me
End Sub

Private Sub lblDemo_DblClick()
  'Restaurar el formato de la etiqueta
  lblDemo.Font.Bold = False
  lblDemo.Font.Italic = False
  lblDemo.Font.Underline = False
  lblDemo.Font.Strikethrough = False
  'Eliminar las marcas de las opciones del menú
  mnuNegrita.Checked = False
  mnuCursiva.Checked = False
  mnuSubrayado.Checked = False
  mnuTachado.Checked = False
End Sub

Private Sub lblDemo_MouseDown(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
          'Si se pulsó botón derecho
  If Button = vbRightButton Then
    PopupMenu mnuContexto, , , , mnuRestaurar
  End If
End Sub

Private Sub mnuCursiva_Click()
  lblDemo.Font.Italic = Not lblDemo.Font.Italic
  mnuCursiva.Checked = Not mnuCursiva.Checked
End Sub

Private Sub mnuNegrita_Click()
  lblDemo.Font.Bold = Not lblDemo.Font.Bold
  mnuNegrita.Checked = Not mnuNegrita.Checked
End Sub

Private Sub mnuRestaurar_Click()
  lblDemo_DblClick
End Sub

Private Sub mnuSubrayado_Click()
  lblDemo.Font.Underline = Not lblDemo.Font.Underline
  mnuSubrayado.Checked = Not mnuSubrayado.Checked
End Sub

Private Sub mnuTachado_Click()
  lblDemo.Font.Strikethrough = Not lblDemo.Font.Strikethrough
  mnuTachado.Checked = Not mnuTachado.Checked
End Sub
```

5. Salvar el formulario en el archivo CONTEXTO.FRM y el proyecto en el archivo CONTEXTO.VBP.

CUADRO 3: EJEMPLO ILUSTRATIVO DE LOS EVENTOS *INITILIZE* Y *TERMINATE*

1. Añadir al proyecto un módulo de clase, y al formulario por defecto dos botones de comando.

2. Establecer las siguientes propiedades en tiempo de diseño:

Form1	*Name*	*frmPrueba*
Caption	*Ejemplo de Initialize y Terminate*	
Command1	*Name*	*cmdCrear*
Caption	*&Crear*	
Command2	*Name*	*cmdDestruir*
Caption	*&Destruir*	
Enabled	*False*	
Clase1	*Name*	*Prueba*

3. Escribir las siguientes declaraciones y procedimientos de evento para el módulo de clase:

```
Option Explicit
Dim Name As String

Private Sub Class_Initialize()
MsgBox "Se ha creado un objeto de prueba"
End Sub

Private Sub Class_Terminate()
MsgBox "Se ha destruido el objeto de prueba"
End Sub
```

4. Escribir las siguientes declaraciones y procedimiento de evento para el módulo de formulario:

```
Option Explicit
Dim Pr As New Prueba

Private Sub cmdCrear_Click()
Pr.Name = "Prueba1"
cmdCrear.Enabled = False
cmdDestruir.Enabled = True
End Sub

Private Sub cmdDestruir_Click()
Set Pr = Nothing
cmdDestruir.Enabled = False
cmdCrear.Enabled = True
End Sub
```

5. Salvar el formulario en el archivo INITER.FRM, el módulo de clase en PRUEBA.CLS y el proyecto en INITER.VBP.

definitivamente el control, debe utilizarse la sentencia *Unload*, ya que ésta permite descargar el control de la memoria. Su sintaxis es la siguiente:

Unload
[Formulario.]MatrizMenús(Índice)

El *cuadro 1* muestra un ejemplo que ilustra la adición y eliminación de opciones de menú en tiempo de diseño. Se trata de un menú en el que se pueden ir añadiendo opciones dinámicamente. Para ello, debe hacerse clic en el botón *Añadir* una vez escrito el título en un cuadro de texto. Además, al hacer clic en el botón *Eliminar*, la última opción incorporada es eliminada.

La segunda opción del menú es una barra separadora que, a su vez, constituye el elemento 0 de la matriz de menús *mnuOpciones*. Según se van añadiendo elementos, estos van tomando los índices 1, 2, 3, etc. El número de opciones del menú se controla mediante una variable declarada a nivel de formulario, denominada *Opciones*. Al seleccionar una de estas opciones en el menú se despliega un mensaje de aviso, indicando el nombre de la misma.

MENÚS EMERGENTES

Los menús que han sido estudiados hasta ahora permiten desplegar una serie de opciones desde la barra de menús. Por ello, se les suele denominar *Menús desplegables*. Sin embargo, existe otro tipo de menús, llamados *Menús emergentes* (*Pop-up menus*), que permiten desplegar sus opciones en cualquier parte del formulario.

En Windows'95, cuando se pulsa el botón secundario del ratón se despliega el menú de contexto correspondiente. En general, la mayoría de las aplicaciones Windows utilizan el botón derecho del ratón para desplegar los menús emergentes, incluido el propio entorno de desarrollo de Visual Basic.

Los menús emergentes se activan en tiempo de ejecución mediante un método especial del formulario, denominado *PopupMenu*. Con él, es posible visualizar cualquier menú o submenú del formulario, siempre que éste contenga opciones. La sintaxis del método es la siguiente:

[Formulario.]PopupMenu NombreMenú[, Indicadores[, X[, Y[, Negrita]]]]

Como puede verse, el método admite tres argumentos opcionales además del nombre del menú. Los argumentos *X* e *Y*, representan las coordenadas donde se desea visualizar el menú, y vienen referidas al área de cliente del formulario. Si se omiten, el menú se visualizará a partir de la posición actual del puntero del ratón.

El argumento *Indicadores* permite especificar una serie de características del menú desplegable, relativas tanto a su localización como a su comportamiento. La tabla 2 muestra los valores posibles de este argumento y su significado.

Si en la llamada se omite el argumento *Indicadores*, se asume el valor 0. Además, los valores relativos a la posición y el comportamiento pueden combinarse mediante el operador *Or*, con el fin de especificar los dos tipos de características.

El argumento *Negrita* permite indicar el nombre de un control de menú cuyo título se mostrará en negrita. Esta opción sólo es tenida en cuenta bajo Windows'95, siendo ignorada en versiones anteriores.

La definición de un menú emergente se realiza de la misma forma que

los menús desplegables de la barra de menús del formulario. Sin embargo, si no se desea que el menú emergente también se muestre sobre dicha barra, será necesario poner a *False* su propiedad *Visible*. Al desplegar el menú emergente, éste siempre será visualizado, independientemente del estado actual de la propiedad *Visible*.

Los menús emergentes suelen visualizarse cuando el usuario pulsa el botón derecho del ratón. Para ello, pueden utilizarse los eventos *MouseUp* o *MouseDown* del formulario. Estos eventos se producen cuando el usuario pulsa y libera un botón del ratón, respectivamente. Su funcionamiento será descrito con mayor detalle en próximos artículos.

El cuadro 2 muestra un ejemplo que ilustra la utilización de menús emergentes como menús de contexto de Windows'95. Se trata de una etiqueta cuyo formato de texto puede cambiarse mediante las distintas opciones de un menú emergente.

El menú emergente se despliega al hacer clic con el botón secundario del ratón sobre la etiqueta. La opción negrita del menú restaura el formato original, y puede ejecutarse también con un doble clic sobre la etiqueta.

BARRAS DE HERRAMIENTAS

Las *barras de herramientas* son conjuntos de controles que proporcionan un acceso rápido a las opciones de menú utilizadas con mayor frecuencia. Son muchas las aplicaciones Windows que utilizan barras de herramientas como método abreviado para facilitar el trabajo del usuario, incluyendo el propio Visual Basic.

Aunque las ediciones Profesional y Empresarial de Visual Basic disponen de un control personalizado, denominado *ToolBar*, que permite crear barras de herramientas de forma realmente sencilla, también es posible crearlas de forma manual, utilizando controles estándar. Para ello, los controles que permiten realizar las acciones desde la barra de herramientas deben situarse dentro de un contenedor que disponga de la propiedad *Align*. Dentro de los controles estándar de Visual Basic, sólo los cuadros de dibujo disponen de esta propiedad.

Cuando la propiedad *Align* del contenedor se pone a 1 (*Arriba*) o 2 (*Abajo*), el control se sitúa en la parte superior o inferior del formulario, ocupando todo el ancho del área de cliente del formulario. Igualmente, cuando se pone a 3 (*Izquierda*) o 4 (*Derecha*), el control se sitúa en la parte derecha o izquierda del formulario, ocupando todo el alto del área de cliente del formulario.

Por tanto, es posible situar barras de herramientas en cualquiera de los bordes del formulario. Si coexisten barras de herramientas horizontales y verticales, las primeras serán las que ocupen el espacio hasta la esquina del formulario. Además, es posible situar varias barras de herramientas del mismo tipo, ya que éstas se van situando una al lado de la otra.

Una vez diseñado el contenedor de la barra de herramientas, deberán situarse dentro de él los controles que se desean mostrar. Normalmente se utilizan botones de comando y cuadros de imagen, aunque también es posible incluir listas desplegables, botones de opción, etc.

La utilización de cuadros de imagen permite simular botones con un dibujo icónico en su interior, describiendo de forma gráfica la tarea que éste realiza. Para ello, debe especificarse en su propiedad *Picture* el archivo con el mapa de bits o el icono correspondiente. En el subdirectorio *BITMAPS* de Visual Basic se incluyen multitud de archivos de mapas de bits especialmente diseñados para la creación de barras de herramientas. La figura 6 muestra un formulario con una barra de herramientas de este tipo.

Cuando los controles de la barra de herramientas realizan la misma operación que una opción de menú, su procedimiento de evento *Click* simplemente tendrá que llamar al de la opción de menú correspondiente. Sin embargo, también es posible crear un procedimiento general que realice la tarea. En este último caso, tanto el procedimiento de evento del control de menú como el del control de la barra de herramientas realizarán la misma llamada.

Las aplicaciones Windows también suelen utilizar otro tipo de barras, denominadas barras de estado, que suelen ser utilizadas para mostrar información acerca del estado actual de la aplicación. Normalmente, este tipo de barras sólo suele incluir controles para visualizar información (etiquetas, imágenes, etc). Sin embargo, su creación es muy similar a la de las barras de herramientas.

CONCLUSIÓN

La utilización de sistemas de menús y barras de herramientas permite dotar a las aplicaciones de un aspecto profesional, que además facilita al usuario la tarea del acceso a las distintas opciones. En el próximo artículo se analizará la forma de crear cuadros de diálogo y manejarlos de forma eficiente.

¿SABE LO QUE SE PIERDE SI NO REGISTRA SUS APLICACIONES PARA MICROSOFT® WINDOWS 95?

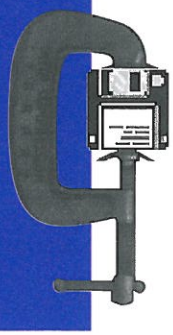
✓ Soporte técnico gratuito por tiempo limitado.
✓ Suscripción gratuita a la revista de los Usuarios de productos Microsoft.
✓ Ofertas en actualizaciones, eventos, seminarios, cursos, etc.
✓ Tener la seguridad de que sus programas de software son legales.

Envíe ya su tarjeta de registro.
Para más información llámenos al telf.: (91) 804 00 96

Office Standard

Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?

PC MEDIA



BORLAND C++ 5.01

Fernando de la Villa

Borland ha lanzado recientemente la versión 5.01 de su conocido compilador Borland C++ 5.01. Esta nueva versión incluye interesantes características que le perfilan como uno de los mejores del mercado. En el presente artículo se trata la edición Developer Suite & Design Tools del mismo.

La nueva versión 5 del compilador de Borland incluye nuevas características y posibilidades para el desarrollador de aplicaciones Windows. Todavía se mantiene la posibilidad de crear aplicaciones para DOS, pero solamente en modo 16 bits. Al fin y al cabo este último es un entorno condenado a desaparecer en un corto espacio de tiempo.

Existen tres versiones diferentes del compilador, cada una con más características que la anterior. En estas páginas se examina la edición completa, denominada Borland C++ 5.01 Developer Suite & Design Tools.

CARACTERÍSTICAS

La edición comentada en el presente artículo incluye todas las características del producto Borland C++ 5.01 Development Suite más la herramienta *Together/C++*. Esta herramienta de análisis y diseño orientado a objetos que proporciona a los desarrolladores una manera cómoda y eficaz de modelar los objetos de un programa, además de generar código. *Together/C++* está formado por tres herramientas que permiten el diseño visual de los modelos:

- *Scenario View Editor*: permite a los programadores realizar una planificación basada en el tiempo de la interacción entre objetos. La finalidad de este editor es facilitar la creación de modelos más completos y, por tanto, diseños más robustos.
- *Object Model Editor*: con el editor de modelos de objetos se puede visualizar toda la información referente a una aplicación. Por ejemplo, se tiene acceso a las clases, atributos, asociaciones, operaciones y relaciones de herencia. Una de las mayores ventajas de este editor es que, al estar la información necesaria contenida en el propio código de la aplicación, no se necesita una base de datos adicional. Además, permite trabajar a la vez sobre el código y sobre el editor, cambiando de uno al otro tantas veces como sea necesario. Este editor soporta las tres notaciones de orientación a objetos más utilizadas (Coad, OMT y Unified).
- *Information Export*: se trata de una utilidad capaz de generar documentación de forma automática, basándose en el código fuente. Cuenta con plantillas predefinidas para los formatos *Rich Text File* (RTF), hipertexto (HTML) y ayuda de Windows (HLP), aunque se pueden definir plantillas para crear documentación en cualquier formato.

La edición *Development Suite 5.01* incluye una serie de herramientas complementarias, además del propio compilador Borland C++. La primera de ellas es *CodeGuard*, un sistema de detección de errores (bugs) en los programas. También se dispone del gestor de ver-

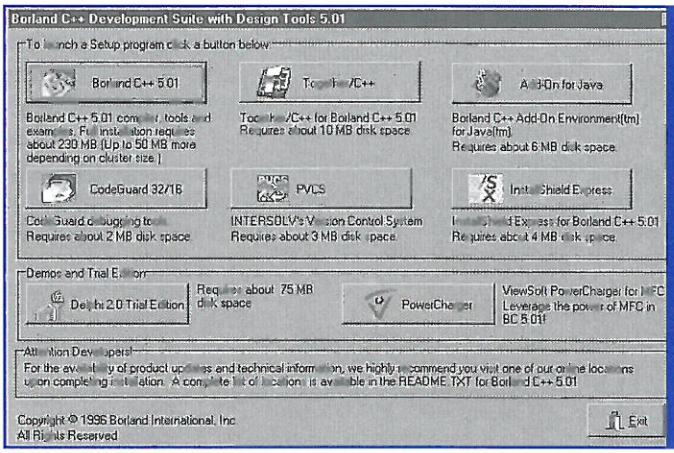

Borland C++ Development Suite with Design Tools 5.01
Borland C++ 5.01
Add-On for Java
Borland C++ Add-On Environment(tm) for Java(tm)
Requires about 6 MB disk space
CodeGuard 32/16
PVCS
Requires about 2 MB disk space
Requires about 3 MB disk space
Demos and Trial Edition
Delphi 2.0 Trial Edition
PowerCharger
Copyright © 1996 Borland International, Inc.
All Rights Reserved
Exit

Figura 1: Together/C++ es una herramienta de análisis y diseño orientado a objetos que ayuda a realizar el modelado de los objetos de un programa.

siones PVCS que se integra perfectamente en el entorno de desarrollo del compilador. Otra herramienta es el conocido *InstallShield Express* cuya utilidad reside en la creación de programas de instalación de forma visual. Por último, se incluye el llamado *AppAccelerator*, un compilador *JIT* (Just In Time) para el lenguaje Java.

Las características de Borland C++ 5.01 incluyen la creación de aplicaciones para DOS, Win32s, y Windows 95, NT y 3.x. Soporta la creación de ejecutables de 16 bits para DOS y Windows, y de 32 bits sólo para Windows. Incorpora un completo entorno (IDE) que centraliza todo el proceso de desarrollo de una aplicación. El IDE proporciona un gestor de proyectos y permite la edición del código fuente, además de integrar casi todas las utilidades, herramientas y expertos incluidos en el producto. También permite la creación de ejecutables para las diferentes plataformas soportadas y facilita la depuración de las aplicaciones. Además, dispone de numerosas librerías de clases, entre las cuales destacan *ObjectWindows* y *ObjectComponents*.

Como regalo, junto a la edición Borland C++ 5.01 Developer Suite & Design Tools se incluye un CD-ROM con la edición Borland C++ 4.5 & DataBase Tools.

LO NUEVO

La versión 5.01 no es más que una actualización de la versión 5.0. Por ello, las diferencias entre ambas son muy pocas. Las nuevas características de la versión 5.01 son las siguientes:

- *OpenHelp*: utilidad para controlar las búsquedas por defecto en los archivos de ayuda.
- Mejoras en la herramienta *Resource Workshop*.
- Soporte para las Visual Database Tools en el experto *ClassExpert*.
- Mejoras en el depurador integrado.

Sin embargo, las mejoras de la versión 5.0 con respecto a la 4.52 sí son numerosas. A continuación se enumeran brevemente las más destacadas:

- Inclusión de las Microsoft Foundation Classes (MFC).
- Nuevo compilador optimizador de 32 bits.
- IDE de 32 bits.

Figura 2: InstallShield es otra de las interesantes herramientas incluidas junto con el compilador de Borland.

- Integración del *Resource Workshop*.
- Nuevas clases *ObjectWindows*: Winsocks, controles comunes, etc.
- El Turbo Debugger para Windows ahora funciona bajo Windows NT.
- Nuevas opciones de línea de comandos para el compilador.
- Inclusión de la librería de plantillas estándar (STL).
- Visual DataBase Tools.
- Nuevas herramientas del kit de desarrollo para Win32.
- Optimización del planificador de Pentium.
- Nuevas opciones para TLINK y TLINK32.
- Depurador de 32 bits integrado con posibilidad de examinar los registros del procesador.

La versión 5.0 incorpora, además, una característica especialmente interesante. Se trata de *ObjectScripting*, un motor de scripts que posibilita la realización de modificaciones al entorno integrado de desarrollo. De este modo se puede adaptar el IDE a las necesidades concretas del desarrollador.

INSTALACIÓN

Para instalar y ejecutar adecuadamente el producto se necesita un ordenador compatible PC que cumpla los siguientes requisitos mínimos:

- 16 megas de memoria RAM (aunque se recomienda un mínimo de 24 megas).
- Procesador 80486 o superior.
- Unidad de CD-ROM.
- 25 megas libres en el disco duro.
- Sistema operativo Windows 95 o Windows NT 3.51.

El espacio libre necesario en el disco duro depende de la instalación que se desee realizar. Existen tres tipos de instalación posible: ejecutar desde CD-ROM, completa y personalizada. Si se elige la opción personalizada la instalación puede ocupar entre 25 y 260 megas. Debido al tamaño de los *clusters* del disco duro en particular, el tamaño puede aumentar considerablemente. Es aconsejable disponer de, al menos, 300 megas libres en el caso de instalar todo el producto.

El programa de instalación permite copiar los archivos sobre una versión anterior del compilador, pero se trata de una copia ciega. Para evitar problemas, se recomienda realizar la instalación sobre un nuevo directorio.

OBJECTSCRIPTING

Lo mejor del IDE incluido en el Borland C++ 5.01 es que puede ser personalizado al gusto del usuario. Esto es posible gracias al motor *ObjectScripting*, que interpreta un lenguaje denominado *cScript*. Este lenguaje para scripts proporciona orientación a objetos y dispone de una semántica y sintaxis similares a las de C++. La programación con *cScript* resulta ser muy sencilla, especialmente para los programadores de C o C++ experimentados. Algunas de las operaciones que se pueden realizar mediante *ObjectScripting* incluyen la grabación automática, el formateo automático del código fuente, la creación de expertos personalizados y la inserción de comentarios de forma automática.

El lenguaje permite controlar los aspectos principales del IDE, como el editor, el gestor de proyectos, el depurador y el buffer de edición. Los programas cScript se pueden crear fácilmen-

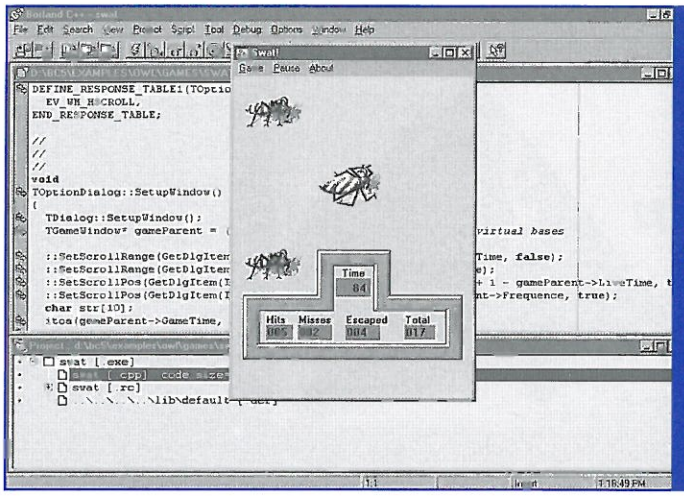


Figura 3: Borland C++ incluye multitud de programas y proyectos de ejemplo, como este sencillo juego para Windows.

te. Basta con utilizar un editor de texto ASCII cualquiera. Para activar un script tan sólo es necesario cargarlo en cualquier momento mediante un comando del menú del IDE.

Además, los resultados de la ejecución de un script pueden verse en la ventana de mensajes del IDE. La información disponible incluye los mensajes de error, de aviso, de diagnosis y también las sentencias de impresión (print).

LOS EXPERTOS

Con el objeto de simplificar la tarea del programador de aplicaciones se han incluido dos expertos: *AppExpert* y *ClassExpert*. Un experto es un programa, normalmente integrado en el IDE, que guía al usuario por los pasos necesarios para realizar una tarea determinada.

AppExpert sirve para generar el código básico de una aplicación Windows, dejando al programador la creación de los detalles concretos. El uso de este experto es muy sencillo. En primer lugar se seleccionan las opciones adecuadas y a continuación se pulsa el botón que activa el proceso de generación automática de código. En el mismo se crean archivos de recurso y archivos de texto enriquecido (RTF), además de los archivos de código fuente. Las aplicaciones creadas con *AppExpert* pueden disponer de opciones de edición multifichero, opciones de impresión, búsquedas, ayuda de contexto, ventana "acerca de...", barra de menú, barra de herramientas y línea de estado.

Las opciones de *AppExpert* están organizadas en tres apartados diferentes. Son:

- *Application*: permite especificar las opciones de las que dispondrá la aplicación. Por ejemplo, línea de estado, lista de archivos usados recientemente y capacidad arrastrar y soltar, entre otras muchas. También permite seleccionar entre una aplicación SDI (Single Document Interface) y MDI (Multiple Document Interface).
- *Main Window*: se utiliza para establecer aspectos relacionados con la ventana principal de la aplicación. Algunos de los parámetros que se pueden fijar en este apartado son el título de la ventana, el color de fondo, la existencia de botones de maximizar y minimizar, y si existe la posibilidad de redimensionar la ventana.
- *MDI Child/View*: desde aquí se selecciona la clase MDI por defecto y los nombres de los archivos fuente necesarios. Por supuesto, sólo es útil para aplicaciones MDI.

Por otro lado, *ClassExpert* es el experto encargado del mantenimiento de clases. Dispone de opciones para crear clases nuevas, editar las clases existentes y navegar por el código fuente. Sólo se puede utilizar con aplicaciones creadas mediante *AppExpert*.

ClassExpert se compone de tres ventanas principales: Classes, Events y la ventana de código. En la primera se muestran las clases derivadas de *ObjectWindows* existentes en la aplicación. Pulsando con el ratón sobre las mismas se pueden realizar diversas acciones como mostrar el constructor de la clase, acceder al último elemento editado y ver la información disponible acerca de la clase. La segunda ventana presenta una lista de los eventos que pueden ser utilizados con la clase seleccionada. Los eventos están ordenados en diferentes categorías como notificaciones de control, funciones virtuales, notificaciones de comandos y mensajes de ventana. Estas categorías están divididas en apartados que ayudan a tener bien organizados los eventos y facilitan su búsqueda. Por último, la ventana de código es en realidad un editor de textos tipo BRIEF. En esta ventana se puede editar cómodamente el código seleccionado, aparte de situar puntos de ruptura para la depuración posterior de la aplicación.

LAS HERRAMIENTAS

Como todo buen compilador de C y C++, Borland C++ 5.01 viene acompañado de una serie de herramientas muy útiles para el desarrollador. A continuación se tratan las características más importantes de cada una de ellas.

InstallShield Express es una herramienta de generación de programas de instalación. Los programadores experimentados conocen las dificultades que entraña la creación de un buen programa de instalación y agradecerán, sin duda, la inclusión de esta herramienta en el producto. *InstallShield* proporciona un método cómodo, sencillo y estándar para la copia de archivos al disco duro, la actualización del registro de Windows y la creación de las carpetas e iconos. Además, se trata de un programa bastante utilizado por las empresas de software y por tanto su interfaz es conocido por la mayoría de usuarios de Windows.

InstallShield dispone de un sencillo interfaz para diseñar el aspecto visual del programa de instalación. La especificación de los archivos y componentes se realiza cómodamente. Incluye 13 cajas de diálogo predefinidas y permite la creación de distintos tipos de instalación. Otra característica interesante es la inclusión de un desinstalador automático de aplicaciones. También soporta compresión de archivos y permite separar la distribución en varios disquetes.

El gestor de versiones PVCS sirve para llevar un control conveniente sobre los cambios realizados en los archivos de una aplicación. Toda la información se almacena en un archivo especial gestionado por PVCS. Los componentes de una aplicación se añaden a este archivo y cada vez que se

desea realizar una modificación sobre alguno de ellos se saca del archivo, se cambia, y se vuelve a introducir en el archivo como una revisión del anterior.

PVCS almacena la historia completa de la aplicación desde su creación. De este modo se mantiene una lista completa y actualizada de los cambios realizados en las diferentes versiones. La información almacenada incluye el nombre de los autores de las modificaciones, los cambios realizados y en qué fecha sucedieron. Soporta el bloqueo de archivos para evitar la inconsistencia de información y permite etiquetas de identificación de versiones.

PVCS se puede instalar, como las demás herramientas, de forma opcional. Si se instala se integra perfectamente en el entorno de desarrollo del compilador. Para utilizar las funciones de la herramienta se utilizan cuatro opciones disponibles en el menú Project.

Otra de las herramientas es *CodeGuard 32/16* cuya utilidad consiste en la detección automática de errores de memoria en las aplicaciones. Sólo se puede utilizar con aplicaciones Windows tanto de 16 como de 32 bits. Al igual que PVCS, se integra sin problemas en el entorno de desarrollo. Para activar *CodeGuard* en una aplicación tan sólo se necesita seleccionar una opción a la hora de definir la plataforma destino del ejecutable de la aplicación. Cuando más tarde se detecte un error, la aplicación deja de ejecutarse y se muestra una ventana con información detallada acerca del inconveniente encontrado. Existe la posibilidad de ir directamente a la línea de código culpable del problema.

CodeGuard muestra información detallada de los problemas encontrados. Esta información incluye, entre otras, el número de bytes escritos en memoria, y el tamaño del bloque de memoria y su dirección. Es capaz de detectar errores de acceso, de recurso, excepciones del sistema operativo y fallos de función. Algunos de los posibles errores de acceso son los desbordamientos de memoria, los accesos a bloques de memoria liberados y los accesos a variables de pila sin inicializar. Los errores de recurso se refieren a problemas con la librería en tiempo de ejecución. Incluyen la detección de desbordamientos de recursos, discordancia de tipos y parámetros incorrectos. *CodeGuard* no puede detectar las excepciones de sistema operativo antes de que se produzcan, pero sí puede mostrar información referente al problema en la ventana de mensajes del IDE y en los archivos de registro. Por ejemplo, soporta las excepciones de división por cero, fallo de página, fallo de protección general, desbordamiento y error del coprocesador.

Otras características de *CodeGuard* incluyen la verificación de punteros, el trabajo con aplicaciones multihilo y la validación de la aritmética de punteros.

Además dispone de una utilidad de configuración que define las opciones de funcionamiento de la herramienta. Esta configuración se almacena en un archivo especial situado en el mismo directorio de la aplicación en desarrollo.

Posiblemente las herramientas más interesantes para los desarrolladores más sofisticados son las referentes a Java. Este lenguaje está arrasando por muchas causas, entre las que destacan su potencia y su capacidad multiplataforma. En esta edición del compilador Borland C++ la capacidad de desarrollo de aplicaciones Java se ofrece como una de las opciones de instalación, aunque se integra en el IDE de forma automática. Permite la creación tanto de applets Java como de aplicaciones completas. El soporte del lenguaje incluye la gestión de proyectos Java, un depurador visual y acceso al compilador de Java.

Los componentes del conjunto de herramientas de Java son:

- *AppExpert for Java*: similar al experto de generación de aplicaciones C y C++, permite la creación de applets Java mediante una simple selección de opciones y también genera una página HTML que haga uso del *applet*.
- *Project Manager*: el gestor de proyectos contiene toda la información relativa a un proyecto y constituye una forma cómoda y adecuada de creación de aplicaciones Java. Entre otras cosas, permite la combinación de archivos C o C++ con Java dentro de un mismo proyecto.
- *TargetExpert*: sirve para configurar correctamente el entorno para la creación de aplicaciones Java. Con este experto se posibilita la creación rápida de proyectos Java.
- *AppAccelerator for Java*: se trata de un compilador Just in Time que aumenta el rendimiento de las aplicaciones Java en la máquina de desarrollo. Con *AppAccelerator* se puede alcanzar un rendimiento hasta 10 veces superior.
- *Visual Debugger for Java*: un depurador especializado en Java. Soporta puntos de ruptura, visualización de variables y ejecución paso a paso.

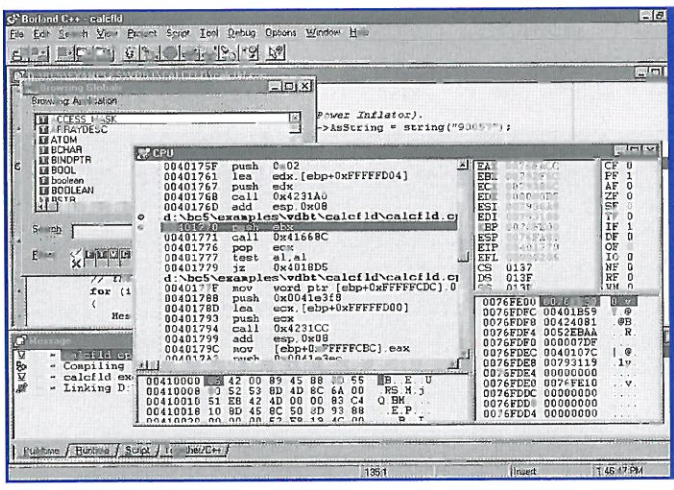


Figura 4: El depurador integrado es un elemento imprescindible de toda buena herramienta de desarrollo.

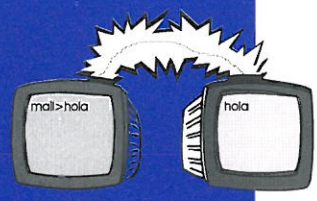
COMENTARIOS FINALES

Borland C++ 5.01 Developer Suite & Design Tools es una herramienta de desarrollo muy completa que proporciona un entorno muy adecuado para la creación de aplicaciones C, C++ y Java. Se nota que el compilador ha evolucionado a la par que las duras exigencias del mercado y se ha adaptado perfectamente a las necesidades de los desarrolladores actuales.

En definitiva, una buena herramienta con unas características nada despreciables y un precio competitivo.

PROTOCOLOS DE COMUNICACIONES: IPX/SPX (2ª PARTE)

María Jesus Recio



En el artículo anterior se hizo una pequeña introducción a la filosofía de trabajo de una red con *Netware* de *Novell*. A modo de recuerdo, cabe destacar entre otras cosas las siguientes:

- La filosofía de trabajo del sistema de red *Netware* de *Novell*: este sistema está orientado al diseño de una red con una estructura cliente-servidor, en la que los ordenadores de la red únicamente tienen acceso a las máquinas definidas como servidores, sin tener la posibilidad de ver el resto de las máquinas clientes.
- La posibilidad de trabajar de forma simultánea con varias pilas de protocolos: recuerde que inicialmente sólo se podía cargar una pila de protocolos en una máquina, por lo que si se quería conectar a otra máquina que funcionaba con una familia de protocolos diferente, era necesario inicializar el ordenador. En la actualidad, este problema está perfectamente solventado con los interfaces *ODI* (interfaz abierta de enlace de datos) de la casa *Novell* y *NDIS* (especificación del interfaz del controlador de red), que permiten manipular de forma paralela diferentes pilas de protocolos.
- La enumeración de las principales características de los protocolos *IPX* (protocolo que trabaja a nivel de red) y *SPX* (que trabaja a nivel de transporte).

Los protocolos *IPX/SPX* están especialmente diseñados para obtener el mejor rendimiento de una red con el sistema operativo *Netware* de *Novell*. Aunque este sistema puede funcionar con otras pilas de protocolos, por ejemplo, *TCP/IP*, el diseño de los protocolos *IPX/SPX* permiten obtener mejores prestaciones de la red.

Los ficheros de configuración básicos de un servidor Netware son startup.cnf y autoexec.cnf

- La definición de una dirección *IPX*, así como sus diferencias con una dirección *IP*. Recuérdese que la dirección *IPX* estaba compuesta por tres partes: dirección de red (que identifica la red o el segmento de la

TABLA 1 CONTROLADORES A UTILIZAR SEGÚN EL TIPO DE DISCOS

Arquitectura del ordenador	Disco	Controlador a carga
ISA	AT	ISADISK
	IDE	IDE
MICROCHANNEL	ESDI	PS2ESDI
	IBM SCSI	
EISA	Clase AT	ISADISK
	IDE	IDE

red en el que está contenido el ordenador), la dirección de nodo, que coincidía con la dirección de la tarjeta de red (por tanto no podía modificarse) y la dirección de *socket* que identifica el proceso hacia el que va dirigido el paquete de información.

- La estructuración de una trama *IPX* y *SPX*, los campos que contenían, así como la encapsulación de una en otra.

INSTALACIÓN DE UN SERVIDOR DE FICHEROS DE NETWARE DE NOVELL

A la hora de instalar un servidor de ficheros de *Netware*, es importante disponer de una serie de datos, que se solicitarán durante el proceso de instalación, como pueden ser el tipo de disco duro que se tiene, el tipo de CD-ROM, etc. En el manual de instalación de *Netware* aparece una hoja similar a la de la figura 1 que debe ser rellenada antes de empezar a instalar.

La tabla 1 muestra los controladores de disco que deben seleccionarse dependiendo del tipo de disco que tenga el ordenador en el que se va a realizar la instalación.

A partir de la versión 4.10 de *Netware* no es necesario crear una partición en el disco duro del ordenador que va a hacer de servidor de ficheros (siempre que se realice una instalación simple del servidor). Esta partición se realiza automáticamente en el proceso de instalación. Dicho proceso utiliza todo el espacio disponible en el disco para realizar la partición *Netware*.

Es necesario señalar que la partición DOS debe ser de al menos 15 Mb. El tamaño de la partición *Netware* dependerá de los módulos y aplicaciones que después quieran instalarse.

Debe entenderse como espacio disponible todo aquel que no forme parte de una partición ya utilizada.

Si el usuario no quiere que el servidor de *Netware* ocupe todo este espacio, debe crear una partición del tamaño deseado antes de empezar con el proceso de instalación.

Un resumen de los pasos que se deben seguir para instalar un servidor es el siguiente:

FIGURA 1

Nombre del servidor:_____ **Nº de red interna IPX**_____
Marca/modelo del servidor:_____ **Nombre del arbol**_____

Tipo de servidor horario_____Zona horaria_____Desplaz del UTC____adelante
Horario de verano_____
_____atrás

Memoria (RAM): Base_____Extendida_____Total_____

Método de arranque del servidor: Disco duro Disquete flexible: 3,5pulg 5,25pulg

Tarjetas de red (Rellene las columnas que se aplican a cada tarjeta de red).

NOMBRE	DRIVER LAN	PUERTO DE E/S	DIRECCIÓN DE MEMORIA	IRQ	DMA	DIRECCIÓN DE NODO	Nº DE RANURA	Nº DE RED EXTERNA

Otras tarjetas (Controladores de discos, adaptadores de video, etc.).

NOMBRE	DRIVER	PUERTO DE E/S	DIRECCIÓN DE MEMORIA	IRQ	DMA	DIRECCIÓN SCSI	INFORMACION ADICIONAL

Discos

MARCA/MODELO	TAMAÑO	DUPLIC. CON EL Nº	SEGMENTOS DE VOLUMEN
1.			
2.			
3.			
4.			
5.			

Volúmenes

NOMBRE	COMPRESIÓN Act FIC. Inact	SUBBLO Act Inac	MIGRACION Act Inac	ESPACIO DE VOL.

Muestra de la hoja a rellenar antes de proceder a la instalación de un servidor *Netware*.

1. Rellenar la hoja descrita.

2. Instalar el *driver* y configurar la tarjeta de red. Aunque la instalación del *driver* de la tarjeta de red puede realizarse durante el proceso de instalación del servidor, es conveniente hacerlo antes; de esta forma, cuando se comience la instalación del servidor, el proceso detectará la tarjeta de red sin ningún problema.

3. Empezar con el proceso de instalación. Si se ha rellenado correctamente la hoja citada, cuando durante la instalación se hagan preguntas sobre controladores a cargar, será muy sencillo contestarlas, pues todos los datos aparecen en dicha hoja.

En el proceso de instalación se solicitan datos como:

- Idioma en el que se quiere realizar la instalación.
- Controlador de disco que se quiera cargar.
- Otros controladores para otros dispositivos, por ejemplo CD_ROM. Cabe destacar que la versión 4.10 de *Netware* soporta un grupo limitado de CD-ROM, por lo que si el del ordenador en el que se está instalando no aparece en este grupo, una vez terminada la instalación dicho dispositivo no va a poder manejarse de una forma cómoda. Este problema ha sido solucionado en la versión 4.11.
- Tipo de tarjeta de red que se tiene instalada, con sus correspondientes datos de configuración: *IRQ, DMA*, etc.
- Clave para el administrador del sistema. Es muy importante dar una clave y recordarla. Un servidor de ficheros de Netware 4.1 no ofrece la posibilidad de entrar al sistema si no se conoce la clave, por lo que si ésta se olvida habrá que reinstalar de nuevo el servidor.
- Nombre que se le quiere dar al servidor, así como el nombre de la organización (este concepto está relacionado con el *NDS* que ofrece esta versión de *Netware*).
- Zona horaria.
- Otras preguntas, como si se quieren instalar las librerías de ayuda.

Nótese que no se solicita el tamaño de la partición de disco para el servidor, pues, como ya se ha dicho antes, esto se hace automáticamente en el proceso de instalación.

CONSIDERACIONES A TENER EN CUENTA

Si la red en la que se está instalando el servidor ya tiene otro servidor de ficheros *Netware*, tiene que cuidarse la dirección interna de red que cada servidor tiene asociada, así como la dirección de nodo.

LISTADO 1
EJEMPLO DEL FICHERO STARTUP.NCF

```
SET RESERVERD BUFFERS VELOW 16 MEG=200
LOAD KEYB España
;cargar del controlador de disco, con la
interrupción y el la dirección de puerto asignados.
LOAD ISADISK.DSK INT=5  PORT=1F0
```

Mientras que la dirección de red interna debe ser la misma en los dos servidores, pues en otro caso entraría en conflicto ya que detectándose el uno al otro (y por tanto suponiendo que están dentro de la misma red) su dirección de red es diferente, en el caso de la dirección de nodo es evidente que ambos deben tener una dirección diferente; si no fuese así, habría conflicto entre quien recibe los paquetes que llevan esa dirección de nodo.

La dirección de red y la dirección de nodo se especifican en el fichero *autoexec.ncf*.

El atributo que define la dirección de red es *NET*; y el atributo que define la dirección del nodo es *NODE_ADDRESS*.

LISTADO 2
EJEMPLO DEL FICHERO AUTOEXEC.NCF

```
set Time Zone = NET-1MEST
set Daylight Savings Time Offset = 1:00:00
set Start Of Daylight Savings Time = (MARCH
SUNDAY LAST 2:00:00)
set End Of Daylight Savings Time = (SEPTEMBER
SUNDAY LAST 2:00:00)
set Default Time Server Type = SINGLE
;nombre del contexto inicial
set Bindery Context = O=CEFAT
; nombre del servidor de ficheros.
file server name ESCUELA
;dirección de red interna
ipx internal net 32987B62
;configuración del driver de la tarjeta de red, junto
a la trama elegida.
LOAD NE2000 INT=5 PORT=300
FRAME=Ethernet_802_2  NAME=NE2000_1_E83
BIND IPX NE2000_1_E83 NET=763B7AF4
LOAD NE2000 INT=5  PORT=300
FRAME=Ethernet_802_3 NAME=NE2000_1_E82
BIND IPX NE2000_1_E83 NET=EA755F11
;montaje de todos los sistemas.
mount all
```

FICHEROS DE CONFIGURACIÓN

Existen varios ficheros de configuración en el servidor de ficheros de *Netware*. Los más importantes son:

- *Autoexec.ncf*
- *Startup.ncf*

Como ya se ha dicho, al instalar un servidor de *Netware* se crea una partición en el disco duro, además de un subdirectorio en la partición DOS (por defecto *nwserver*) con algunos ficheros de arranque y configuración de *Netware*.

FICHERO STARTUP.NCF

El fichero *startup.ncf* se encuentra en la partición DOS y se ejecuta cuando se está cargando el servidor de *Netware*; por tanto se trata de un fichero de arranque del sistema. Contendrá, por tanto, órdenes que permitan inicializar el sistema, como son: órdenes para cargar controladores de disco, órdenes para cargar espacios de nombres, órdenes set para mejorar el rendimiento del sistema. El listado 1 muestra un ejemplo posible de este fichero.

FICHERO AUTOEXEC.NCF

Este fichero se encuentra localizado en la partición *Netware*. Su ejecución se lleva a cabo después de la carga del sistema, utilizándose, entre otras cosas, para:

- Cargar módulos.
- Definir la configuración del sistema *Netware*.
- Montar sistemas de ficheros.
- Almacenar datos referentes a la red, como son: número *IPX* de red interna, nombre del servidor de ficheros, definiciones horarias, número de nodo, controladores de las tarjetas de red, etc.

El listado 2 muestra un ejemplo de este fichero.

OTROS FICHEROS

Aunque lo habitual es que el servidor de ficheros *Netware* tenga y trabaje con los ficheros vistos, a veces un servidor

puede utilizar otros ficheros de configuración como es el fichero *NETINFO.CFG*. Este fichero almacena los comandos asociados a la configuración de protocolos, siempre que se haya utilizado para ello la utilidad *INETCFG*. A diferencia de los ficheros citados anteriormente, este fichero nunca debe editarse; su manipulación debe realizarse siempre a través del comando *inetcfg*.

Los ficheros de arranque y configuración de un cliente Netware son dos: startnet.bat y net.cfg

La manipulación de estos ficheros se puede llevar a cabo de dos formas:

1. Cargando el módulo editor: si se escribe:

load edit sys:system\autoexec.ncf
load edit c:\nwserver\startup.ncf

se editará el fichero correspondiente, permitiéndose realizar todas las modificaciones oportunas.

2. A través de la orden *install*: al ejecutar la orden:

load install

se entrará en una utilidad que permite reinstalar nuevos módulos y controladores, además de ofrecer la posibilidad de editar los ficheros de configuración; para hacerlo se debe:

Seleccionar la opción Opciones de archivos *NCF*.
Editar el fichero *(autoexec.ncf/ startup.ncf)*

Nótese que el fichero *startup.ncf*, al ubicarse en la partición DOS, es posible editarlo directamente con *edit*, antes de cargar el servidor de *Netware*.

SERVIDOR DEDICADO

Es bueno recordar que un servidor de ficheros de *Netware* es un servidor dedicado, es decir, se trata de una máquina en la que no "puede hacerse nada", ya que se dedica a prestar servicios a los clientes de la red que se conectan a ella.

En realidad sí se pueden realizar algunas operaciones de control del sistema. Existen unos 75 comandos y alguna utilidad que pueden ejecutarse en el servidor. Se trata de comandos y utilidades que permiten modificar elementos de la configuración del sistema, cargar nuevos módulos, descargar módulos, montar dispositivos y volúmenes, ver información acerca de las direcciones de red y protocolos, cambiar algún parámetro seleccionado durante la instalación, etc.

Lo que no se puede hacer cuando el servidor está arrancado como tal, es ejecutar las aplicaciones destinadas a los clientes, ni nada que se encuentre en la partición DOS del disco.

También pueden cargarse otros servidores dentro del propio servidor de ficheros, como por ejemplo un servidor de impresora, o uno de comunicaciones.

Si se quieren realizar modificaciones en el servidor de ficheros es necesario ejecutar la orden install

MODIFICACIONES EN LA INSTALACIÓN

Como ya se ha dicho, si se quieren realizar modificaciones en el servidor de ficheros, una vez instalado, es necesario ejecutar la orden *install (load install)*. Al hacerlo, aparecerá un menú en el que se puede seleccionar el tipo de opciones que desean modificarse, como por ejemplo:

- Opciones del controlador (Cargar/Descargar controladores).
- Opciones de disco (Configurar/Duplicar/Probar partición).
- Opciones de volumen (Configurar/Montar/Desmontar volúmenes).
- Opción de Licencia (Instalar la Licencia del servidor).
- Opciones del directorio (Instalación del *DS*).
- Opciones de archivos *NCF* (Crear/Editar archivos de inicialización).
- Opciones de productos (Otros elementos optativos de instalación).
- Opciones del servidor (Instalación/actualización de este servidor).

Cada una de estas opciones llevan a un nuevo menú donde se puede elegir entre diversas operaciones relacionadas, que dependerán de la opción previa elegida.

Leyendo las opciones puede deducirse que desde aquí se puede modificar completamente la instalación inicial del servidor, ya que el conjunto de opciones cubre la totalidad de los elementos del sistema.

INSTALACIÓN DE UNA ESTACIÓN CLIENTE DE NETWARE DE NOVELL.

Una vez vista la instalación de un servidor de ficheros *Netware*, es necesario describir como se instala la parte cliente en un ordenador.

Lo primero que se debe tener claro es el cliente que se quiere cargar, es decir, si se quiere cargar únicamente un cliente para DOS, o bien si se desea cargar un cliente para Windows.

En el caso de Windows, si se trata de Windows 95, y se está instalando la versión 4.10 de *Netware*, se debe elegir entre cargar un cliente que trabaja en modo real (el que ofrece *Netware* de

Novell) o si se carga un cliente que trabaja en modo protegido (el ofrecido por el propio Windows 95). En el artículo sólo se explicará cómo cargar el cliente *Netware* ofrecido por *Novell* para Windows 3.1 y Windows 3.11, ya que el cliente para Windows 95 se instala como el resto de los clientes de otros protocolos en Windows 95.

Los disquetes que contienen el cliente de *Netware* se pueden generar en el servidor de ficheros de *Netware*, ya que están contenidos en el propio CD-ROM del sistema operativo. Para generarlos lo que se debe hacer es:

1. Insertar el CD-ROM del sistema operativo.
2. Cargar la utilidad *install*, con la orden *load install*.
3. Seleccionar la opción Opciones de productos.
4. Aparece un nuevo menú, seleccionar la opción Creación de disquetes de instalación de cliente DOS/MS Windows/OS2.
5. Seleccionar el cliente y seguir las instrucciones del sistema.

De esta forma se generarán cuatro disquetes con el cliente *Netware* que posteriormente se utilizarán en todos los ordenadores de la red que quieran trabajar como clientes *Netware*.

Después de generar los disquetes del cliente y una vez elegido el cliente a cargar, los pasos que deben darse para su instalación son los siguientes:

Inicialmente, y antes de empezar con la instalación del cliente propiamente dicho, es necesario configurar e instalar el controlador de la tarjeta de red. Para ello deben utilizarse los disquetes que el fabricante de la tarjeta de red incluye. Es muy importante apuntar la *IRQ* y la dirección de puerto que se asignan a la tarjeta, para posteriormente hacer coincidir estos valores con los valores solicitados durante la instalación del cliente *Netware*.

La *IRQ* en una tarjeta de red (en otro tipo de tarjetas también) puede modificarse vía software o vía hardware, dependiendo de la tarjeta.

LISTADO 3
EJEMPLO DE UN FICHERO STARTNET.BAT

```
set NwLanguaje = español
rem Módulo que hace de interfaz entre los controladores de red y los protocolos de rem comunicaciones, como IPX.
c:\nwclient\LSL.COM
   rem Se trata del controlador de la tarjeta de red.
c:\nwclient\NE2000.COM
   rem Se trata del módulo para ODI.
c:\nwclient\IPXODI.COM
   rem Programa que permite la comunicación con el servidor.
c:\nwclient\.VLMCOM
```

Evidentemente, de forma independiente al método de asignación de este parámetro, su valor, así como el valor de la dirección de puerto, nunca debe coincidir con los valores asignados a otros dispositivos del ordenador, ya que si es así ambos dispositivos entrarán en conflicto y el sistema dejará de funcionar correctamente. Por tanto, antes de configurar la tarjeta de red, es necesario comprobar los valores de los que se dispone para estas dos características. Lo habitual es asignar la *IRQ* 5 ó 10, y la dirección de puerto 300, siempre y cuando estén libres.

Una vez configurada la tarjeta de red, se puede empezar con el proceso de instalación del cliente *Netware*. Para ello, deben seguirse estos pasos:

- Ejecutar la orden *install* contenida en el primer disquete del cliente.
- El sistema preguntará el directorio en el que se va a instalar el cliente, la autorización para modificar los ficheros *autoexec.bat* y *config.sys* del sistema, el directorio en el que se encuentra Windows (cuando se quiere instalar el cliente para DOS y para Windows), y la tarjeta de red de la que dispone la máquina, que será detectada si previamente se ha instalado.

A partir de aquí el proceso irá solicitando disquetes hasta terminar la instalación del cliente.

Las modificaciones introducidas en los ficheros de arranque de DOS, son las siguientes:

- Fichero *config.sys*: introduce las siguientes líneas:
  LASTDRIVE=Z
  FILES=40
- Fichero *autoexec.bat*: introduce la siguiente línea:
  @CALL STARTNET.BAT
- además de modificar la variable *PATH* en la que introduce el directorio seleccionado para la instalación del cliente, y que por defecto es: *c:\nwclient*.

FICHEROS DE CONFIGURACION

El cliente de *Netware*, por defecto se instala en el directorio *c:\nwclient*; en él aparecerán algunos ficheros de configuración del cliente, así como los módulos de carga del cliente, y otras utilidades.

FICHERO STARTNET.BAT

El fichero *startnet.bat* contiene los módulos que se necesitan para cargar el cliente de *Netware*. Aunque en el artículo anterior ya se mostró un ejemplo del fichero, se hace necesario introducir de nuevo el ejemplo, que aparece en el listado 3 de este artículo.

FICHERO NET.CFG

Este fichero también se encuentra en el directorio que contiene el cliente *Netware*; se trata de un fichero similar

LISTADO 4
EJEMPLO DE UN FICHERO NET.CFG

```
; valores asociados por Netware a la tarjeta de red
Link driver NE2000
        PORT 300
        IRQ 10
        FRAME ETHERNET_802.2
;valores asociados al cliente Netware.
Netware DOS Requester
        FIRST NETWORK DRIVE = F
        NETWARE PROTOCOL = NDS BIND
;Contexto por defecto en el que buscará el usuario
        NAME CONTEXT = EDUCACION.CEFAT
```

al *config.sys* del DOS. Contiene valores de configuración que son leídos e interpretados al iniciar el cliente *Netware*. El listado 4 muestra un ejemplo de este fichero.

ARRANQUE

Una vez instalado el cliente *Netware*, basta ejecutarlo; como la línea que hace la llamada al fichero *startnet.bat* está incluida en el fichero *autoexec.bat* del sistema, al arrancar la máquina, automáticamente se cargará lo necesario para poder conectarse al servidor.

Para desconectarse del sistema basta con escribir la orden *logout*, al hacerlo el sistema se despide dando información sobre la hora de conexión y desconexión.

MAPEO DE DIRECTORIOS

Mapear un directorio consiste en asignar una letra, como si se tratase de una letra de unidad a un directorio concreto.

El cliente *Netware* mapea de forma automática todos los directorios contenidos en la variable *PATH* del DOS como unidades lógicas, además de mapear algunos directorios del servidor (aquellos que contengan órdenes *Netware*). De esta forma y mientras se esté conectado al servidor, se va a tener acceso a todos los comandos, ya sean DOS o *Netware*, sin necesidad de especificar el camino completo de su localización.

La IRQ en una tarjeta de red puede modificarse vía software o vía hardware, dependiendo de la tarjeta

Después de la ejecución del fichero *startnet.bat*, basta con hacer un *login* con un nombre de usuario para estar dentro del servidor. Para hacer un *login* al servidor, los pasos que deben darse son los siguientes:

- Cambiarse a la unidad F.
- Ejecutar la orden *login*.
- Introducir el nombre de la cuenta y la contraseña asociada. Si todo se escribe correctamente, después de esto el usuario se encontrará dentro del sistema.

Téngase en cuenta que el usuario con el que se hace *login* debe aparecer contenido en el contexto especificado en el fichero *net.cfg*. En otro caso, para que el sistema detecte al usuario, es necesario especificar el contexto completo además del nombre del usuario.

Para especificar un contexto, se debe empezar escribiendo el nombre del usuario y a continuación los contenedores en los que está incluido, hasta llegar a la raíz del árbol. Cada uno de estos valores debe aparecer separado del anterior y del posterior por el carácter punto. Un ejemplo de especificación de *login* es el siguiente:

coco.educacion.cefat

ORDEN MAP

Esta orden permite realizar dos operaciones muy importantes:

- Mapear los directorios en los que se busca los ejecutables escritos en línea de comando (similar a la variable *PATH* del DOS). A esto se le llama unidades de búsqueda. Se utiliza tanto para los directorios contenidos en la variable *PATH* del DOS, como para directorios del servidor de ficheros de *Netware*. Habitualmente estos mapeos se realizan empezando por la letra F (las letras anteriores en el abecedario se utilizan para las unidades locales de la máquina cliente), y siguiendo en orden directo el abecedario.
- Llamar con letras de unidad a caminos completos. Esto se hace debido a que las rutas en *Netware* son muy largas, ya que deben contener no sólo el camino de directorios, sino también el nombre del servidor y del volumen dentro de este. Si a un *path* completo se le asigna una letra, como si se tratase de una unidad de disco lógica, para cambiarse a él basta con escribir la letra correspondiente. Habitualmente estas letras se asignan empezando por la letra Z, y siguiendo el orden inverso del abecedario.

Otra forma de mapear es utilizar una herramienta de usuario para Windows, incorporada por *Netware* que se denomina *NetwareTools*, y que permite realizar estas operaciones de forma más cómoda y sencilla.

Los mapeos (los correspondientes a los discos del servidor de ficheros *Netware*) deben realizarse en alguna de las secuencias de conexión, para así disponer de ellos cada vez que uno se conecte al sistema.

Tras la ejecución del fichero startnet.bat, basta con hacer un login para estar dentro del servidor

PRÓXIMO NÚMERO

El próximo artículo de esta serie seguirá profundizando en la forma de trabajo del sistema operativo *Netware* de *Novell*, centrándose principalmente en la estructuración que a partir de la versión 4.1 *Novell* ha incorporado a su sistema operativo. Se hablará del *NDS*, de su filosofía orientada a objetos, y del manejo de la principal utilidad incorporada por *Netware* para el control del sistema: *Nwadmin*.

LA ILUMINACIÓN (I)

Pedro Antón Alonso

Este mes se comienza a ver la teoría de los modelos de iluminación, así como un ejemplo práctico de cómo aplicar al rellenado *gouraud* un modelo de iluminación *lambertiana*, observando claramente la mejoría obtenida.

Se recuerda cómo comenzaba el artículo en que se trataba del sombreado *gouraud*: "Los objetos físicos son sólidos era frase con la que comenzaba la anterior entrega. Eso es algo sobre lo que no cabe la menor duda, pero ninguna superficie plana va a recibir, normalmente, igual cantidad de luz en todos sus puntos."

Hasta aquí todo correcto, pero ¿cómo calcular la intensidad de luz que incide en un punto del objeto que estamos modelando?

Para ello, primero estudiaremos los distintos modelos de luz.

LA LUZ AMBIENTAL

Probablemente el modelo de iluminación más sencillo.

En este modelo de iluminación se establece una cantidad de luz ambiental, igual para todos los puntos de la escena, tomando como foco de luz una fuente puntual.

Partiendo de la ecuación básica de iluminación, suponiendo que la luz ambiental afecta de igual forma a todas las superficies desde todas las direcciones, se tiene: $I = Ia.\ k$ donde Ia es la intensidad de luz ambiental que hemos supuesto igual para todos los objetos, y k es el coeficiente de reflexión ambiental, coeficiente que se puede considerar como característica del material de la superficie.

Esta luz de ambiente produce un sombreado plano sobre las caras del objeto, sombreado que no permite distinguir las distintas caras del objeto, como se puede apreciar en la figura 1.

Ahora bien, se puede tratar la reflexión difusa de algunos materiales opacos y mates como reflexión *lambertiana* porque la energía luminosa que se irradia desde cualquier punto de la superficie se rige por la ley del coseno de Lambert. Esta ley establece que, en un material con este tipo de reflexión, la cantidad de luz que refleja cualquier área de superficie pequeña dA hacia el observador es directamente proporcional al coseno del ángulo formado entre la dirección al observador y el vector normal (perpendicular) del plano. Por lo tanto, para las superficies *lambertianas*, la cantidad de luz que ve el observador es independiente y sólo es proporcional al coseno del ángulo de incidencia de la luz.

No obstante, la cantidad de luz que refleja una superficie dependerá de la orientación de la superficie respecto a la fuente de luz. Es decir, si la superficie es perpendicular a la dirección de la luz incidente, brillará con mayor intensidad que si dicha superficie tiene una orientación oblicua con respecto a la fuente de luz (ver figura 2).

EL TIEMPO REAL

Implementar un modelo de iluminación en tiempo real no es tarea fácil.

Vista la teoría, viene la ardua tarea de implementar el modelo en tiempo real.

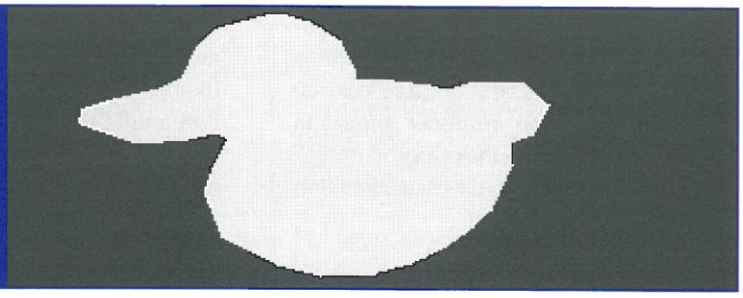
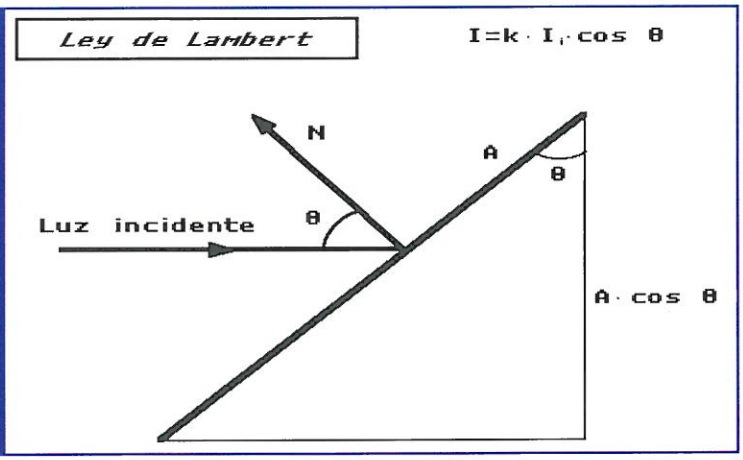


Figura 1

Figura 2



Las máquinas crecen y cada vez se añaden más efectos reales a los programas, una buena muestra de ello es la iluminación. Pero no se debe olvidar que en el mundo de la *demoscene* llevamos ya un par de años observando este modelo implementado en tiempo real; para ello, los programadores han de buscar la forma de ahorrar el mayor número posible de cálculos, manteniendo una apariencia del modelo expuesto aceptable.

A continuación se estudiará el modelo de iluminación *lambertiana* junto con los medios a nuestro alcance, es decir, disponemos de un rellenado *gouraud* que como parámetros de entrada acepta coordenadas y color de una paleta de 256 colores.

CÁLCULO DE LA NORMAL DE UN VÉRTICE

Calcular el vector normal de un plano es sencillo, pero ¿cómo calcular el vector normal de un vértice?

Si la rutina de rellenado *gouraud* tiene como parámetro de entrada, entre otros, el color de cada uno de los vértices, no bastará con saber la intensidad de color que incide sobre el plano, sino que se deberá conocer la cantidad de luz que incide en cada vértice. Hasta el momento esa cantidad de luz estaba siendo calculada mediante la profundidad del objeto, pero evidentemente el resultado no es suficientemente bueno.

Para calcular el vector normal de un vértice de un objeto dentro de una malla, calcularemos, en primer lugar, el vector normal de cada una de las caras que conforman el objeto. Para lo cual, como era de esperar, se usará el producto vectorial.

Recordemos que el producto vectorial de dos vectores es otro vector de módulo, el producto de los módulos por el seno del ángulo que forman, sentido dado por la regla del sacacorchos y dirección perpendicular al plano formado por ambos vectores (ver figura 3). Esto último nos da la solución a la primera parte del problema. Todo esto debe ser ya conocido por el lector, ya que el sentido de este producto es el que nos indica la visibilidad o no de un plano, como se explicó en la segunda entrega de la parte dedicada a las 3D de este curso, y en ello está basada la función visible de la unidad *LIB3DFP.PAS*, desarrollada a lo largo del curso.

Figura 4

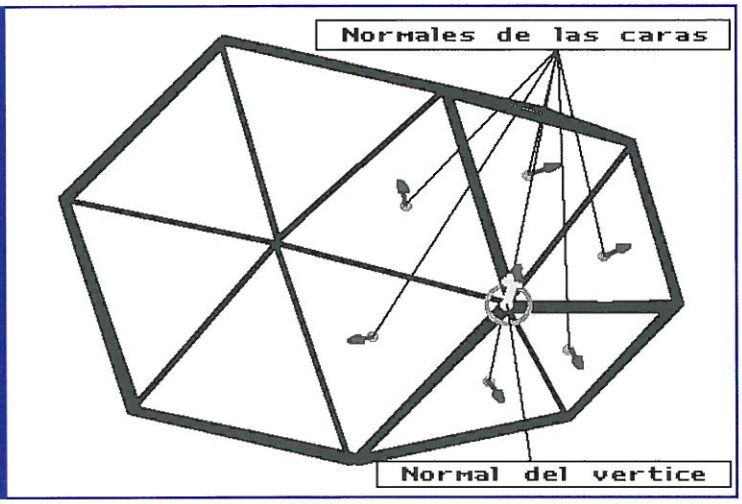
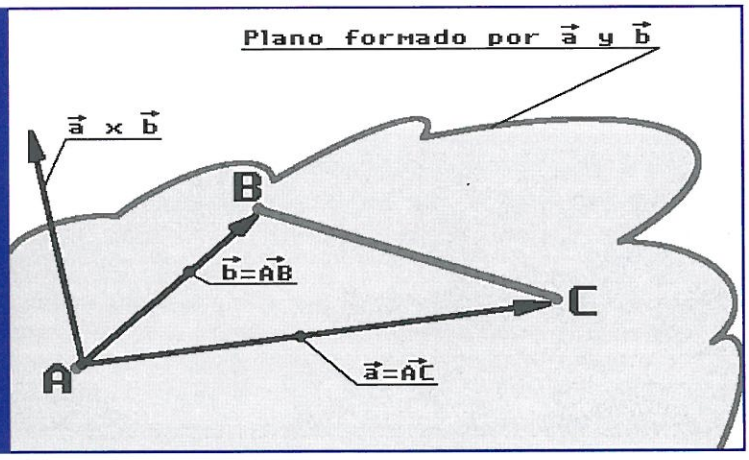


Figura 3



Una vez calculados todos los vectores normales de cada una de las caras, el vector normal de cada vértice será la media aritmética de los vectores normales de las caras coincidentes con dicho vértice. Como se observa en la figura 4.

De este proceso se encarga la función *CalcNormals*. El lector puede imaginar que este proceso será lento, pero no debe preocuparnos, ya que dichos valores son precalculados, es decir, sólo se calculan una vez, no están dentro del bucle principal.

EL CÓDIGO

En el Listado 1 se incluye la parte de la unidad *LID3DFP* que ha sido creada en este artículo, donde se pueden observar las funciones que a continuación se comentarán.

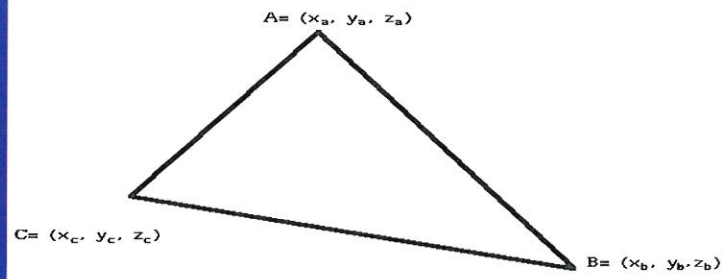


Figura 5

$$\vec{AB} = (x_b - x_a,\ y_b - y_a,\ z_b - z_a) = (x_1,\ y_1,\ z_1)$$
$$\vec{AC} = (x_c - x_a,\ y_b - y_a,\ z_b - z_a) = (x_2,\ y_2,\ z_2)$$
$$\vec{AB} \times \vec{AC} = \begin{vmatrix} x & y & z \\ x_b - x_a & y_b - y_a & z_b - z_a \\ x_c - x_a & y_c - y_a & z_c - z_a \end{vmatrix}$$

Coordenada Z de $\vec{AB} \times \vec{AC}$: $(x_1\ y_2) - (x_2\ y_1)$

La unidad *LIB3DFP* para la tabla de cosenos punto fijo 8.8, luego parece evidente que podemos encontrar alguna ventaja usando esos valores para normalizar nuestros vectores; también resulta lógico ampliar los tipos *TPoint3D* y *TFace3D* para almacenar los vectores.

Veamos cómo trabaja la función *CalcNormals*. Como se acaba de comentar, en primer lugar, debemos calcular el vector normal de cada plano, para lo cual realizamos un sencillo producto vectorial (ver figura 5) eligiendo los vectores formados por los puntos AB y AC, dichos vectores son el resultado de restar las coordenadas (x,y,z) del destino al origen. Lo más peculiar de esta función es el paso siguiente. Calcular el módulo del vector normal y normalizar dicho vector, para a continuación multiplicarle por el valor *_NORMALIZE*, valor constante e igual a 256. Recordemos que para normalizar un vector basta con dividir cada una de sus componentes por su módulo, quedando los valores del vector comprendidos entre 0 y 1. La función *CalcNormals* fija esos valores entre 0 y 255.

Una vez hecho esto, se comprueba vértice a vértice qué planos son los que coinciden con cada vértice, para sumar sus valores normales y dividir entre el número de planos coincidentes, de esta forma se calcula el vector normal de cada vértice. Estos datos son almacenados en los nuevos campos *nx,ny,nz* de los registros *TPoint3D* y *TFace3D*, como se aprecia al comienzo del listado 1.

LA LUZ

Se ha creado un nuevo tipo *TLight3D*, que contiene los siguientes campos:

x1,y1,z1 : Origen de la luz.
x2,y2,z2 : Destino de la luz.
ux,uy,uz : Vector de dirección de la luz.
I : Intensidad de la luz.

Se podría crear el tipo más reducido; de hecho el campo intensidad de luz no se usa en esta entrega, pero he considerado más oportuno crear más campos de los estrictamente necesarios, para aumentar la claridad del artículo. El hecho de que una luz ambiental y puntual (que irradia intensidad luminosa en todas direcciones de igual forma) apunte a algún sitio no resulta de fácil comprensión, ¿verdad?. La respuesta la veremos más adelante.

Asociado a este tipo existen dos funciones *InitLight* y *KillLight*. Recordemos que al usar memoria virtual para la definición de nuestras variables, hemos de solicitarla y liberarla; precisamente de eso y de algo más se encargan estas funciones.

Pero en el listado 1 se puede ver que la función *InitLight* hace algo más y tiene un parámetro de entrada *scale*. Pues bien, inicialmente este tipo de luces no están pensadas para generar asombrosos efectos de luminosidad, sino para dar un resultado final mucho más real a la escena que estamos creando. De esta forma, colocar una luz en un sitio más o menos adecuado y normalizar su vector de dirección nos va a ahorrar muchos cálculos. Esta inicialización, un tanto especial, también es realizada por la función *InitLight*.

EL EJEMPLO

En el ejemplo que, como es habitual, acompaña al artículo se pone en práctica todo lo expuesto en estas líneas, obteniendo el resultado final observado en la figura 6. El tema es bastante complejo, y el autor recomienda la bibliografía del final del artículo a aquellos lectores a los que les haya sabido a poco lo expuesto aquí.

En primer lugar se comentará la función *VectorsAngle* incluida en el listado 1. Esta función es el alma del efecto, ya que se encarga de obtener los colores que van a ser pasados a la rutina de rellenado *gouraud*. Como se ha dicho al principio del artículo la Ley de Lambert nos dice que la intensidad de luz para un modelo de reflexión *lambertiana* es igual a una constante del material, que varía entre 0 y 1, y tomaremos como 1, para ahorrar cálculos, por la intensidad de luz, que evidentemente también tomaremos como 1, por el coseno del ángulo que forman. Pero si los dos primeros datos son la unidad, nos bastará con el coseno del ángulo que forman, quedando nuestro problema realmente reducido.

El problema ahora reside en calcular el coseno del ángulo formado por el vector de luz, y el vector normal al plano que vamos a rellenar, que previamente hemos calculado. Esto se realiza aplicando el producto escalar de dos vectores, pero antes es necesario hablar de las modificaciones incluidas a la función *Calc3DRotations*, encargada de rotar los puntos en el espacio. Ahora esta función, además de rotar estos puntos, rotará también las normales de cada

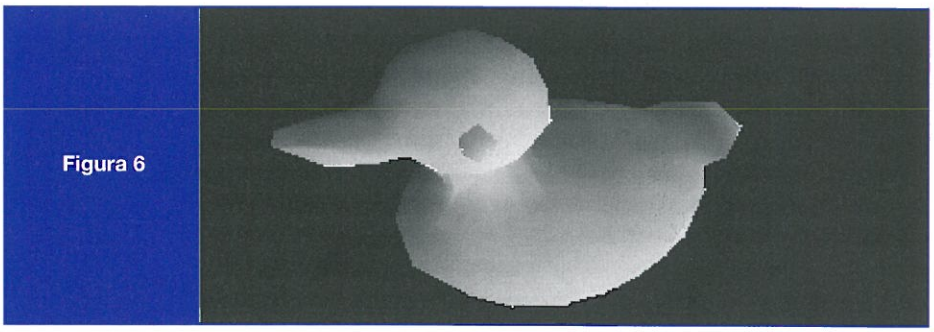


Figura 6

LISTADO 1

```
UNIT Lib3DFP;
TYPE
 TPoint3D   = record
              x,y,z    : longint; { 3D coor.      }
              nx,ny,nz : longint; { Normal vector.}
             end;
 PPoint3D   = ^TPoint3D;
 TLight3D   = record
              x1,y1,z1 : longint; { from...       }
              x2,y2,z2 : longint; { ... to, then  }
              vx,vy,vz : longint; { vector!       }
              I        : longint; { Intensity     }
             end;
 PLight3D   = ^TLight3D;
 TFace3D    = record
              a,b,c    : dword;   { Face def.     }
              nx,ny,nz : longint; { Normal vector.}
              z        : longint; { for QuickSort.}
             end;
 PFace3D    = ^TFace3D;

CONST
 _NORMALIZE: longint =  256;

VAR
 Light      : PLight3D;

PROCEDURE CalcNormals    (ObjPoints : PPoints;
                          ObjFaces  : PFaces;
                          NumPoints , NumFaces : dword);
PROCEDURE InitLight      (scale     : integer);
PROCEDURE KillLight;
FUNCTION  LambertLight   (vx1,vy1,vz1: longint;
                          vx2,vy2,vz2: longint ):longint;
IMPLEMENTATION

PROCEDURE  CalcNormals  (ObjPoints:PPoints;
ObjFaces:PFaces;NumPoints,NumFaces:dword);
var
  CntFaces   : dword;
  CntPoints  : dword;
  Module     : longint;
  vx1,vy1,vz1 : longint;
  vx2,vy2,vz2 : longint;
  nx,ny,nz   : longint;
  CheckPoints : longint;
begin
{ First we are going to calculate the Normal Vector
for each face... }
{  MaxValue:=0; { ... normalize vectors! }
  for CntFaces:=0 to NumFaces do
    begin
      vx1:=ObjPoints^[ObjFaces^[CntFaces].b].x-
           ObjPoints^[ObjFaces^[CntFaces].a].x;
      vy1:=ObjPoints^[ObjFaces^[CntFaces].b].y-
           ObjPoints^[ObjFaces^[CntFaces].a].y;
      vz1:=ObjPoints^[ObjFaces^[CntFaces].b].z-
           ObjPoints^[ObjFaces^[CntFaces].a].z;
      vx2:=ObjPoints^[ObjFaces^[CntFaces].c].x-
           ObjPoints^[ObjFaces^[CntFaces].a].x;
      vy2:=ObjPoints^[ObjFaces^[CntFaces].c].y-
           ObjPoints^[ObjFaces^[CntFaces].a].y;
      vz2:=ObjPoints^[ObjFaces^[CntFaces].c].z-
           ObjPoints^[ObjFaces^[CntFaces].a].z;
     nx:=(vy1*vz2)-(vz1*vy2);{ Only is a cross product
}
      ny:=(vz1*vx2)-(vx1*vz2);
      nz:=(vx1*vy2)-(vy1*vx2);
      Module := round (sqrt ((nx*nx)+(ny*ny)+(nz*nz)));
      if Module>0 then { avoid divide by zero }
        begin
          ObjFaces^[CntFaces].nx:=(nx*_NORMALIZE)
div Module;
          ObjFaces^[CntFaces].ny:=(ny*_NORMALIZE)
div Module;
          ObjFaces^[CntFaces].nz:=(nz*_NORMALIZE)
div Module;
        end
      else
        begin
          ObjFaces^[CntFaces].nx:=0;
          ObjFaces^[CntFaces].ny:=0;
          ObjFaces^[CntFaces].nz:=0;
        end;
    end;
{ ... and now, we are going to calculate the normal
vector for each point }
  for CntPoints:=0 to NumPoints do
    begin
      nx:=0;
      ny:=0;
      nz:=0;
      CheckPoints:=0;
       for CntFaces:=0 to NumFaces do { Check the
point for ALL faces }
        begin
          if (ObjFaces^[CntFaces].a=CntPoints) or
             (ObjFaces^[CntFaces].b=CntPoints) or
             (ObjFaces^[CntFaces].c=CntPoints) then
            begin
              nx:=nx+ObjFaces^[CntFaces].nx;
              ny:=ny+ObjFaces^[CntFaces].ny;
              nz:=nz+ObjFaces^[CntFaces].nz;
              Inc (CheckPoints);       { CoUnT the num-
ber of points }
            end;
        end;
      if CheckPoints>0 then { avoid divide by zero }
        begin
                    ObjPoints^[CntPoints].nx:=nx  div
CheckPoints;
                    ObjPoints^[CntPoints].ny:=ny  div
CheckPoints;
                    ObjPoints^[CntPoints].nz:=nz  div
CheckPoints;
        end
      else
        begin
          ObjPoints^[CntPoints].nx:=0;
          ObjPoints^[CntPoints].ny:=0;
          ObjPoints^[CntPoints].nz:=0;
        end;
    end;
end;

PROCEDURE InitLight (scale:integer);
var
  Module : longint;
begin
  new (Light);
  Light^.x1:=_NORMALIZE*scale;
  Light^.y1:=_NORMALIZE*scale;
  Light^.z1:=_NORMALIZE*scale;
  Light^.x2:=0; { pointing to center screen }
  Light^.y2:=0;
  Light^.z2:=0;
  Light^.vx:= Light^.x2-Light^.x1;
  Light^.vy:= Light^.y2-Light^.y1;
  Light^.vz:= Light^.z2-Light^.z1;
                                    Module:=round
(sqrt((Light^.vx*Light^.vx)+(Light^.vy*Light^.vy)+(Li
ght^.vz*Light^.vz)));
  if Module>0 then
    begin
          Light^.vx:=  (Light^.vx*_NORMALIZE)  div
Module;
          Light^.vy:=  (Light^.vy*_NORMALIZE)  div
Module;
          Light^.vz:=  (Light^.vz*_NORMALIZE)  div
Module;
    end;
end;

PROCEDURE KillLight;
begin
  dispose (Light);
end;

FUNCTION                                LambertLight
(vx1,vy1,vz1,vx2,vy2,vz2: longint):longint;
var
  DotProduct : longint;
begin
  DotProduct := (vx1*vx2)+(vy1*vy2)+(vz1*vz2);
    asm  mov  eax,[DotProduct];  sar  eax,9;  mov
[DotProduct],eax end;
  VectorsAngle:=127+DotProduct;
end;
```

BIBLIOGRAFÍA

INTRODUCCIÓN A LA GRAFICACIÓN POR COMPUTADOR.
Autores: Foley. van Dam. Feiner. Hughes. Phillips.
Editorial: Addison-Wesley IberoAmericana.

GRÁFICAS POR COMPUTADORA
Autores: Donald Hearn. Pauline Baker.
Editorial: Prentice Hall.

vértice, ya que éstas varían de igual forma que sus respectivos vértices. Volviendo al producto escalar, éste es igual a la suma del producto de cada una de las componentes x, y, z de los vectores, e igual, a su vez, al producto del módulo de ambos vectores por el coseno del ángulo que forman. Conocido el valor de los módulos, aproximadamente igual a *_NORMALIZE*, el problema se simplifica enormemente: ese valor estará comprendido entre -127 y +128 aproximadamente, debido a las normalizaciones previamente realizadas; por lo tanto, bastará dividir el valor entre 512 y sumarle 127, para obtener, de forma realmente rápida, el valor buscado que nos indicaba la ley de Lambert. Dicho valor nos da directamente el valor del color buscado para cada vértice. La figura 6 ilustra el resultado que, como el lector puede comprobar es realmente superior al obtenido dando a cada vértice el color de su profundidad, algo suficiente para salir del paso, pero no demasiado satisfecho.

Y para finalizar comentar cómo se efectúa el movimiento de la luz. Como lo mas importante es conseguir una buena velocidad sin perder, por otro lado, efectividad en el resultado, la luz describe una circunferencia alrededor del eje y de esta forma la luminosidad varía dependiendo del ángulo formado entre el vector de dirección de la luz y la normal de cada vértice, de acuerdo al modelo de iluminación *lambertiana*, expuesto en el artículo, con un mínimo gasto de CPU. Aunque, como notará el lector, en el código que acompaña a esta entrega se están realizando todos los cálculos de rotación y proyección tridimensionales. Algo totalmente absurdo en este ejemplo, ya que la figura permanece estática. El motivo es evidente, comprobar el gasto de CPU añadiendo este modelo de iluminación al bucle principal.

CURSO DE VISUAL C++ 4.0 (XII)

PROGRAMACIÓN DE WINDOWS SOCKETS

Jorge R. Regidor

Desde hace unos años, el mayor avance desarrollado en el entorno informático ha estado relacionado con la interconexión de diferentes ordenadores: lo que todos conocemos como redes. Estas van desde un conjunto de ordenadores interconectados en una oficina, hasta la red de redes con conexiones por todo el mundo, como Internet.

Dentro del mundo de las redes existen varios protocolos de comunicación que permiten el entendimiento entre las máquinas, no sólo entre las del mismo tipo, sino entre las que son diferentes, posibilitando el intercambio de información. Entre estos protocolos se encuentran *NetBeui*, IPX/SPX, XNS o *DECNet*, pero el protocolo que destaca por su utilización sobre todos los demás es el utilizado en Internet, el protocolo TCP/IP, ampliamente estudiado en otras secciones de Sólo Programadores.

En este artículo se va a exponer un método para programar aplicaciones que se comuniquen por un entorno de red mediante los conocidos *Windows Sockets*. No es un estudio de ningún tipo de protocolo de red, simplemente se trata de dar unos conceptos y formas básicos para comunicar con éxito dos aplicaciones por *Windows Sockets* utilizando las clases brindadas por *Visual C++ 4.0*. Por todo esto, se recomienda al lector con pocos o ninguno conocimientos en redes y protocolos que se ilustre brevemente de algunos de los conceptos del protocolo TPC/IP (véanse los artículos del curso de sistemas abiertos de Fernando J. Echevarrieta, números 25, 26 y 27 de Sólo Programadores).

EL PROTOCOLO TCP/IP

El protocolo TPC/*IP (Transmission Control Protocol/Internet Protocol*) fue desarrollado a finales de la década de los años 60 por el departamento de defensa de Estados Unidos. El objetivo final que se pretendía era interconectar diferentes tipos de máquinas y sistemas de forma transparente, independientemente de su tamaño, arquitectura hardware o sistema operativo.

Dentro del protocolo, IP es el encargado de *enrutar* los paquetes con los datos: cada máquina es identificada con un número exclusivo, la llamada dirección IP. Esta dirección se compone de 4 bytes, que se representan como cuatro valores entre 0 y 255 separados por puntos. Además, dentro del protocolo TCP, se definen una serie de puertos que identifican el tipo de aplicación por la que se quiere comunicar. Ejemplo de estos puertos son: el 21 para control de FTP, el 23 para la aplicación *Telnet* o el 80 utilizado para las aplicaciones WEB.

SOCKETS

Un *socket* es un punto final de comunicación que permite a dos aplicaciones de dos máquinas dentro de la red comunicarse. Para ello, un *socket* combina la dirección IP con un puerto determinado, consiguiendo así identificar una aplicación dentro de una máquina. Los *sockets* son bidireccionales, lo que permite que los datos fluyan simultáneamente en ambos sentidos.

Cuando el TCP manda una petición de conexión a otra máquina, incluye el número de identificación del *socket* emisor, compuesto por su dirección IP y por el puerto a utilizar. Si la máquina destino acepta la conexión, junto con la indicación de aceptación devuelve también su dirección IP y el número de puerto a utilizar.

Dos máquinas pueden comunicarse por cualquier puerto, pero existe un convenio por el cual los puertos menores de 255 están reservados. Por ejemplo, el puerto 21 se utiliza para FTP, el

puerto 23 para TELNET, el puerto 25 para SMTP, el puerto 110 para POP3, etc.

WINDOWS Y LOS SOCKETS

Para soportar los *sockets* desde Windows, se creó una arquitectura *WOSA* que permite tener un API totalmente estándar para cualquier tipo de suministrador del *driver* del dispositivo de red; gracias a ello cualquier aplicación que sea compatible con *Windows Socket* funciona con cualquier dispositivo que suministre un *driver* análogo.

Este API está basado en la distribución de la universidad de Berkeley, por lo que las rutinas incluidas dentro del fichero de cabecera *WINSOCK.H* son compatibles con la especificación de los *sockets* definidos por Berkeley.

Las principales funciones de este fichero de cabecera son:

*accept** : Una petición es reconocida y asociada con el *socket* creado inmediatamente. El *socket* original continuará escuchando nuevas peticiones de conexión.
*closesocket** : Cierra el *socket* indicado.
*connect** : Inicia una conexión con el *socket* especificado.
listen: Espera por nuevas peticiones de conexión.
*recv** : Recibe datos del *socket* conectado.
*recvfrom** : Recibe datos de un *socket* conectado o no.
*send**: Manda datos hacia el *socket* conectado.
*sendto**: Manda datos hacia un *socket* conectado o no.
socket: Crea un punto final de comunicación y devuelve el *socket*.

Las funciones con * pueden ser bloqueantes.

Desde luego que éstas no son todas las funciones incluidas dentro del API, pero sí las más representativas y las que dan una idea global antes de pasar un poco más adelante y ver las clases de *Visual C++ 4.0* destinadas a trabajar con *sockets*.

WINSOCKECTS Y LAS MFCS

La clase *CAsynSocket* representa un *Windows Socket*. Un punto de final de comunicaciones por red. Esta clase encapsula todo el *API Windows Socket*, orientándolo hacia objetos y permitiendo una mejor integración con las aplicaciones creadas con MFC.

Quien utiliza esta clase debe ser buen conocedor de las comunicaciones por red, por lo que será responsable de manejar los bloques, detectar las diferencias de direccionamento entre máquinas y las conversiones entre códigos de caracteres. En este *casp* no se va a utilizar esta clase directamente, sino la clase *CSocket* derivada de ésta, bastante más sencilla, sobre todo al integrarse con la clase *CArchive* para realizar los envíos y recepciones de datos como si se escribiese o leyese en un fichero.

LA CLASE CSOCKET

La clase *CSocket* deriva de la clase *CAsynSocket* y se encarga de encapsular el API para los *Windows sockets*. Los métodos de los objetos del tipo *CSocket* son bloqueantes, lo que implica que las funciones como *Send, Receive, ReceiveFrom, SendTo* o *Accept* no devuelvan el control a la aplicación hasta que no finalicen su ejecución.

Después de crear el objeto *CSocket* se debe llamar a la función miembro *Create*, la cual crea el *handle* al objeto del tipo SOCKET existente y encapsulado por esta clase. Por defecto esta función miembro crea un *stream socket*, aunque se le puede indicar que cree un *datagram socket*, así como el número del puerto por el cual se quiere realizar la comunicación. El formato de la función es:

```
BOOL Create( UINT nSocketPort = 0, int
nSocketType = SOCK_STREAM,
  LPCTSTR lpszSocketAddress = NULL );
```

donde,
uSocketPort es el puerto TCP.
nSockettype es el tipo de *socket*. Los tipos pueden ser:

- *SOCK_STREAM*: Es un tipo de *socket* bidireccional, con comprobación de orden y duplicidad.
- *SOCK_DGRAM*: Es un tipo de *socket* bidireccional, sin comprobación de orden y duplicidad. En este tipo de *sockets* es responsabilidad del programador mantener el orden y la no duplicidad de paquetes.

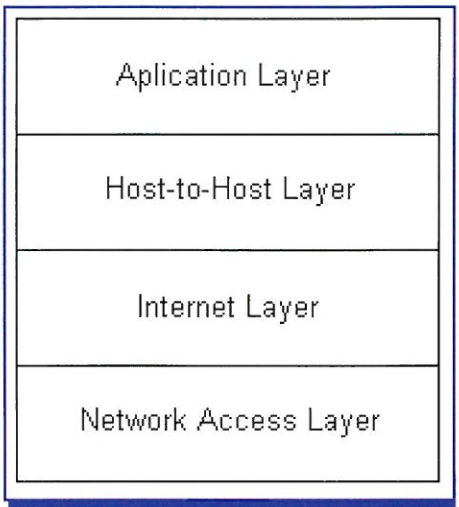


Figura A.
Arquitectura del protocolo TCP/IP

lpszSocketAddress es la dirección de la estructura *SOCKADDR* que contiene la dirección de red.

Hay que destacar que si el tipo de *socket* elegido es *SOCK_DGRAM*, el objeto *CSocket* no puede utilizarse conjuntamente con la clase *CArchive*.

Las operaciones básicas de esta clase son las siguientes:

Accept: Que acepta una petición de conexión de un *socket*.
Close: Cierra el *socket*.
Connect: Conecta el *socket*.
Listen: Pone al objeto a escuchar posibles peticiones de conexión.
Receive: Recibe datos del *socket* conectado.
Send: Envía datos al *socket* conectado.

Además de estas funciones miembros, se pueden sobrecargar las siguientes funciones para atención a mensajes de notificación:

OnAccept: Notifica a un *socket* de escucha que puede aceptar la conexión.
OnClose: Notifica a un *socket* que el *socket* conectado ha cerrado la conexión.
OnConnect: Notifica que el *socket* se ha conectado correctamente.
OnReceive: Notifica que han llegado datos.

OnSend: Notifica que ya puede llamarse de nuevo a la función *Send* para enviar nuevos datos.

Como se puede observar las operaciones básicas para el manejo de esta clase son pocas y conceptualmente muy sencillas, como se confirmará más adelante.

En este artículo se va a describir el método para combinar esta clase junto con la clase encargada de serializar datos sobre ficheros, como es *CArchive*. La relación entre ambas viene dada por la clase *CSocketFile*, que permite enviar y recibir datos por el *socket* como si de un fichero se tratase.

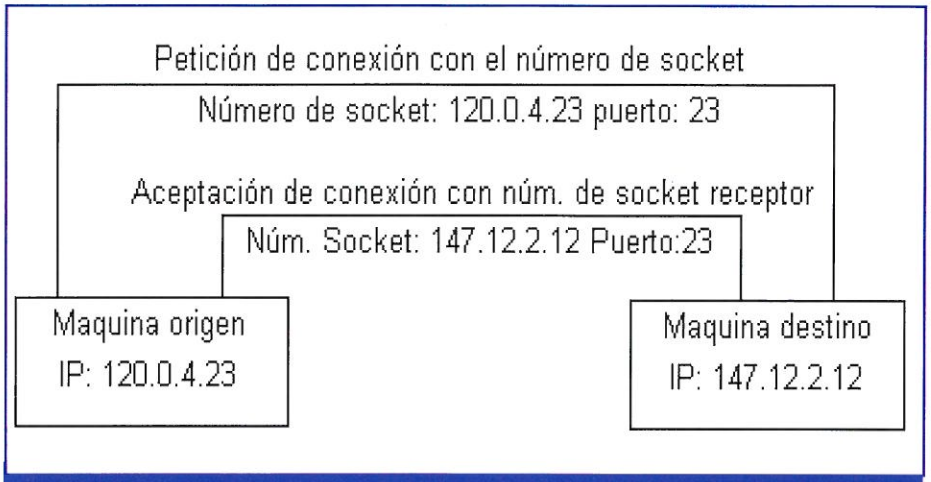

Petición de conexión con el número de socket
Número de socket: 120.0.4.23 puerto: 23
Aceptación de conexión con núm. de socket receptor
Núm. Socket: 147.12.2.12 Puerto:23
Maquina origen
IP: 120.0.4.23
Maquina destino
IP: 147.12.2.12

Figura B. Escenario de conexión de dos máquinas por un socket

LA CLASE CSOCKETFILE

La clase *CSocketFile* se deriva de la clase *CFile* usada para mandar y recibir datos por la red mediante los *Windows Sockets*. Esta clase permite asociar las operaciones de entrada/salida sobre ficheros con transmisiones de datos por la red, lo que conceptualmente es un tanto extraño, pero en realidad facilita mucho la labor de programación.

La asociación entre esta clase y la clase *CSocket* se realiza al llamar al constructor, ya que el primer parámetro del constructor es un puntero al objeto *CSocket*. El constructor de *CSocketFile* tiene el siguiente formato:

CSocketFile(CSocket pSocket, BOOL bArchiveCompatible = TRUE);*

donde:

pSocket es el puntero al objeto *CSocket* a asociar.

bArchiveCompatible indica si el objeto va a ser asociado a otro objeto *CArchive* o bien va a ser utilizado sólo como si fuera un objeto *CFile* con alguna restricción como la imposibilidad de realizar búsquedas.

Normalmente, el hecho de asociar de este modo los dos objetos es lo único que se debe hacer con esta clase, ya que luego se asociará a una clase *CArchive*, la cual va a encargarse de realizar las operaciones de entrada/salida o, mejor dicho, las operaciones de envío y recepción de datos.

LA CLASE CARCHIVE

La clase *CArchive* sirve para guardar objetos en un dispositivo permanente como puede ser un disco duro. Esto hace que posteriormente estos objetos puedan ser recuperados de nuevo por la aplicación. El proceso de hacer que los datos sean permanentes es denominado serialización. Un objeto *CArchive* permite la entrada y salida mediante *buffers* de datos desde o hacia un fichero.

Antes de crear un objeto del tipo *CArchive*, debe estar creado un objeto del tipo *CFile*, para ser asociado con el objeto *CArchive*. En este caso el objeto a asociar es de la clase *CSocketFile*, clase derivada de *CFile*.

Un objeto *CArchive* puede no sólo procesar tipos primitivos, sino objetos de clases derivadas de la clase *CObject* diseñadas para ser "serializadas". Esto se consigue utilizando las macros DECLARE_SERIAL e IMPLEMENT_SERIAL e implementando una función miembro llamada *Serialize* que se encarga de grabar o cargar sus propios datos.

Hay que destacar que, si se utiliza la capacidad de enviar y recibir objetos MFC utilizando el objeto *CArchive* asociado a una clase *CSocketFile*, se debe comunicar con otra aplicación MFC, ya que de lo contrario los datos recibidos no serán debidamente interpretados.

Otra nota importante es la incompatibilidad de los *sockets* basados en datagramas DGRAM_SOCKETS con la clase *CArchive*, por lo que si se quiere utilizar este tipo de *sockets* debe ser mediante la clase *CAsynSocket* y sin estar asociados a la clase *CArchive*. La razón es que los *sockets* basados en datagramas no garantizan ni el orden ni la duplicidad de paquetes, lo que va totalmente en contra del concepto de serialización utilizado en la clase *CArchive*. Si se intenta asociar un *socket* basado en datagramas a una clase *CArchive*, una excepción MFC será lanzada.

EL MODELO DE PROGRAMACIÓN DE LA CLASE CSOCKET

Como ya se ha comentado anteriormente, usar un objeto *CSocket* implica crear y asociar otras dos clases más. Todo este proceso se va a ver en detalle en las siguientes líneas, tanto desde el punto de vista del servidor como desde el del cliente.

Lo primero que hay que comentar es la diferencia entre un *socket* servidor y

El API de Windows Sockets está basado en la distribución de la universidad de Berkeley

uno cliente. Un *socket* servidor está constantemente "escuchando" en un puerto para reconocer posibles peticiones de conexión desde los diferentes clientes. Este caso se puede ilustrar muy bien al ver un servidor WEB que constantemente está escuchando su puerto para ver nuevas peticiones de conexión y de ese modo poderle enviar las páginas adecuadas a petición del cliente.

Los pasos para establecer la conexión entre un *socket* servidor y uno cliente son los siguientes (los pasos son comunes para los dos tipos de *sockets* salvo que se especifique lo contrario):

1. Crear un objeto del tipo *CSocket*.
2. Usar este objeto para crear el *handle* SOCKET encapsulado por la clase. Para crear un *socket* cliente normalmente llamaremos a la función miembro *Create* con los parámetros por defecto, pero para un *socket* servidor se debe especificar el puerto por el cual se van a escuchar las peticiones de conexión.
3. Si el *socket* es cliente se debe llamar a la función miembro *CAsynSocket::Connect* para establecer la conexión con el *socket* servidor. Si el *socket* es el servidor se debe llamar a la función *CAsynSocket::Listen* para atender a las peticiones de conexión. Cuando una petición llegue, se lanzará el evento *OnAccept*, donde se llamará a la función miembro *CAsynSocket::Accept*. Esta función recibe una referencia, un nuevo objeto *CSocket* vacío, por lo que habrá que llamar al constructor y sólo a éste, antes que a *Accept*.
4. Hay que crear el objeto *CSocketFile* y asociarlo con el objeto *CSocket* creado.
5. Hay que crear un objeto *CArchive* para grabar (enviar) y leer (recibir) datos y asociarlo con el objeto *CSocketFile* creado anteriormente.
6. Ya se puede usar el objeto *CArchive* para pasarse datos entre los *sockets*. Normalmente, ya que los objetos *CArchive* o bien sólo pueden grabar o bien sólo pueden cargar datos, será necesario crear dos objetos *CArchive* asociados a *CSocketFile*, uno para grabar (enviar) y otro para cargar (recibir).
7. Por último, sólo queda destruir cada uno de los objetos, comenzando por *CArchive, CSocketFile* y *CSocket*.

Hay que apuntar que dentro de la clase *CArchive* existe una función miembro especialmente diseñada para ser utilizada con la clase *Csocket;* se trata de la función miembro *IsBufferEmpty*. Esta función permite saber, en caso de existir más de un mensaje en el *buffer* de recepción, si éste está vacío o no para leer más datos y dejar vacío el *buffer* de recepción.

Un socket es un punto final de comunicación que permite a dos aplicaciones de dos máquinas dentro de la red comunicarse

SECUENCIA DE OPERACIONES

En el siguiente apartado vamos a ver con pseudocódigo fuente todos los pasos a realizar para establecer la comunicación, primero desde el punto de vista del servidor y luego del cliente:

```
CSocket sockSrvr;
// Construir el socket del servidor
sockSrvr.Create(nPort);
// Crear el socket por el puerto nPort.
SockSrvr.Listen();
// Comenzar a escuchar peticiones de
// conexión.
// Se recibe una petición de conexión.
// Dentro del evento OnAccept
CSocket sockRecv;
// Construir el socket para conectarse al
// que lanzo la petición.
sockSrvr.Accept(sockRecv);
// Aceptar la conexión
CSocketFile file(&sockRecv);
// Construir el fichero basado en un
// socket.
CArchive arIn(&file,CArchive::load);
// Crear un objeto archivo para
// recibir datos.
CArchive  arOut(&file,CArchive::store);
// Crear un objeto archivo para
// enviar datos.

// Para transmitir datos

arIn  >> dwValue;
// Recibir sobre dwValue.
ArOut << dwValue;
// Enviar dwValue.
```

El caso del cliente es bastante parecido salvo que no hay que crear un *socket* para escuchar.

```
CSocket sockClient;
// Construir el socket del cliente
sockClient.Create();
// Crear el socket.
sockClient.Connect(strAddr,nPort);
// Conectar el socket a la máquina
// indicada por la dirección IP de
// strAddr y al puerto nPort.
// Se busca la conexión
CSocketFile file(&sockClient);
// Construir el fichero basado en un
// socket.
CArchive arIn(&file,CArchive::load);
```

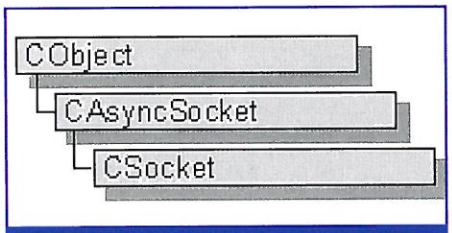


Figura C. Jerarquía de clases relacionadas con sockets

```
// Crear un objeto archivo para
 // recibir datos.
CArchive  arOut(&file,CArchive::store);
// Crear un objeto archivo para
// enviar datos.
// Para transmitir datos

arIn  >> dwValue;
// Recibir sobre dwValue.
ArOut << dwValue;
// Enviar dwValue.
```

Como se puede ver el proceso de conexión de los *sockets* es muy sencillo, y el paso de datos todavía más, ya que se pueden pasar datos de la misma manera que cuando se quieren grabar o cargar desde disco.

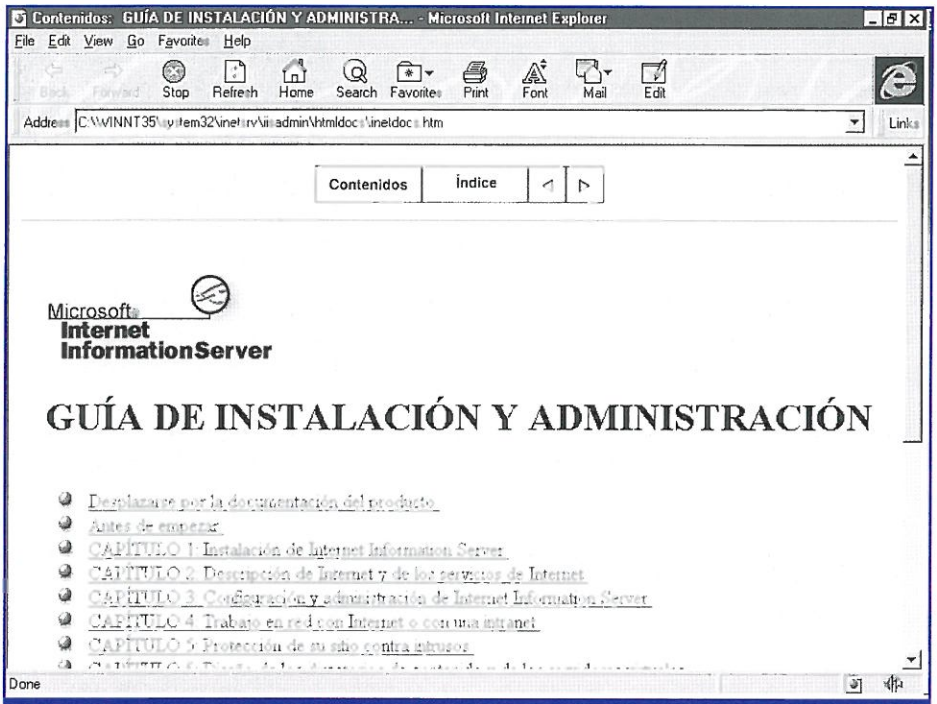

Contenidos: GUÍA DE INSTALACIÓN Y ADMINISTRA... - Microsoft Internet Explorer
File Edit View Go Favorites Help
Stop Refresh Home Search Favorites Print Font Mail Edit
Contenidos
Índice
Microsoft Internet InformationServer
GUÍA DE INSTALACIÓN Y ADMINISTRACIÓN
Done

Figura D. Los clientes WEB utilizan sockets para comunicarse.

PROGRAMA DE EJEMPLO

A modo de ejemplo de programación de *Windows Sockets* con MFC se puede crear un control capaz de reutilizarse en posteriores programas. Es decir, un componente OLE (OCX). El tema de cómo crear este tipo de controles desde *Visual C++ 4.0* se tratará en el siguiente artículo, pero ahora lo que sí se va a ver brevemente son algunas de las especificaciones de este control, así como algunos fragmentos del código cuya comprensión no requiera conocimientos de cómo crear controles OLE.

En primer lugar un control OCX permite reutilizar un código ya programado, y, para ello, acceder a él por medio de las propiedades, los métodos y los eventos. Las propiedades permiten cambiar el comportamiento del control, con los métodos se pueden realizar determinadas acciones, y, con los eventos, recibir de modo asíncrono notificaciones de determinadas cosas que ocurren internamente en el control y de las cuales deberíamos enterarnos.

La implementación del control permitirá crear *socket* clientes, y en consecuencia dispondrá de los siguientes métodos:

Connect: Se encarga de crear el *socket* y lanzar la petición de conexión.

Los sockets basados en datagramas no pueden ser utilizados con la clase CArchive, ya que no garantizan ni el orden ni la duplicidad de los datos

Close: Se encarga de cerrar el socket.

Receive: Se encarga de recibir los datos.

Send: Se encarga de enviar los datos.

Las propiedades necesarias en un primer momento son:

Connected: Que informará si el *socket* está o no conectado.

HostAddress: Dirección IP del *host* al cual está conectado.

HostName: Nombre del *host* al cual está conectado. Para que funcione por nombre debe haber activo un servidor de nombres de dominio(DNS).

Port: Puerto por el cual se pretende comunicar.

Y, por último, los eventos a atender deberían ser:

OnClose: Evento lanzado cuando se recibe una petición de cerrar el *socket*.

OnReceive: Evento lanzado al detectar datos recibidos por el *socket*.

CONCLUSIÓN

En este artículo se ha pretendido dar una visión práctica del modo de programar los *Windows Sockets* desde *Visual C++ 4.0* y su librería de clases MFC, sin entrar en detalles ni de arquitectura de redes, ni del propio protocolo TCP/IP.

Se ha pretendido dar una visión general del método a seguir y mostrar además la sencillez de éste. Se espera que fruto de esto y debido al gran auge de las comunicaciones, las próximas aplicaciones incorporen ya la capacidad de comunicarse mediante este método tan sencillo y tan utilizado entre toda la comunidad de programadores de comunicaciones como son los *sockets*.

SIGUIENTE ARTÍCULO

En el próximo artículo se va a detallar la forma de crear un control OLE (OCX) y para ello se realizará uno que permita hacer comunicaciones por *sockets*. Esto permitirá reutilizar este control desde otras aplicaciones, ya sean *Visual C++ 4.0* , *Visual Basic 4.0* o cualquier otro lenguaje que acepte controles OCX.

Plantear esta forma de crear una aplicación que maneje los *socket* sirve para que un posible control genérico de comunicaciones por *sockets* pueda ser ampliamente reutilizado desde otras aplicaciones, y de paso aprovechar esto para hablar de otro de los pilares de *Visual C++ 4.0*: la posibilidad de crear componentes OLE reutilizables.

Contactar con el autor :
jjrio@arrakis.es

LOS TIPOS DE DATOS (I)

Felipe Bertran y César Sánchez

Este mes se explicará el concepto de tipo de datos, y se verán algunos de los distintos tipos de datos predefinidos en el lenguaje, así como la forma de definir los nuestros propios. Se estudiará el concepto de atributo y se mostrarán algunos de los atributos asociados con los tipos de datos presentados. Por último se introducirán dos estructuras de datos compuestas, los records y los arrays.

En el artículo pasado se dio una pequeña visión histórica del lenguaje Ada y se motivó al lector subrayando la importancia que tiene este lenguaje en el desarrollo de grandes sistemas de software, especialmente en el ámbito de los sistemas empotrados y en sistemas de tiempo real. Se escribió un pequeño programa para mostrar el aspecto que ofrece la sintaxis del lenguaje, así como para ilustrar los procesos de compilación y enlazado con el compilador de libre distribución GNAT. Se introdujeron también varios conceptos básicos generales, como son las variables, las constantes, los procedimientos y las funciones.

En el presente artículo se pretende dar una visión general y algunos detalles acerca de los tipos de datos.

TIPOS

Un tipo se puede definir como un conjunto de valores y un conjunto de operaciones que actúan sobre esos valores. Ya se conocen, de los sencillos programas de la entrega anterior, dos tipos: el tipo Integer y el tipo Float. En el primero, Integer, por ejemplo, el conjunto de valores asociado es el de los números enteros {...,-2,-1,0,1,2,...} y las operaciones son la suma (+), la resta (-), la multiplicación (*), etc.

El mecanismo que provee el lenguaje Ada para la definición de nuevos tipos es lo que se conoce como declaración de tipo. Los tipos predefinidos, como Integer o Float, no son una excepción, y se puede encontrar su declaración de tipo dentro del paquete Standard, que viene con el compilador.

Se ha de indicar que el lenguaje provee automáticamente de las operaciones de asignación (:=) comparación de igualdad (=) y desigualdad (!=) entre objetos de tipos definidos por el usuario.

Antes de ver cómo se escribe una declaración de tipo, es necesario entender qué es un lenguaje fuertemente tipado. En un lenguaje fuertemente tipado, como Ada, cada objeto tiene asociado un tipo; debido a ello tiene acotado claramente el conjunto de valores que puede almacenar y las operaciones que se pueden realizar con ese objeto. Ello impide, por ejemplo, que podamos asignar directamente un valor de punto flotante a una variable de tipo entero, o viceversa. Otros lenguajes, como por ejemplo C, permitirían esa asignación, haciendo una conversión implícita de tipos, probablemente truncando el valor en punto flotante. En Ada se tomó la decisión de no permitir la conversión implícita de tipos por varios motivos. En primer lugar, puede conducir a programas ejecutables con errores muy sutiles y muy difíciles de detectar. Esto es debido a la posible pérdida de información que se produce en la operación de conversión (en el caso anterior, al convertir de Float a Integer se pierden las cifras decimales). En segundo lugar, la posibilidad de mezclar automáticamente tipos en asignaciones y en operaciones conduce a un código desorganizado. Tipos distintos corresponden a conceptos distintos, y por lo tanto no deberían mezclarse. El verdadero peligro no reside en la conversión en sí misma, sino en que ésta se realice de forma implícita,

inconsciente, sin que aparezca reflejada en el código. Por ello Ada sí permite la conversión explícita, de forma que para asignar, por ejemplo, a la variable Integer A, el valor de la variable de tipo Float B, se puede escribir:

A:=Integer(B);

Para ser exactos, hay que decir que Ada redondea (no trunca) el valor decimal de B al entero más cercano en esta conversión.

DECLARACIONES DE TIPOS

Como ya se ha mencionado, Ada permite la definición de nuevos tipos. Ello solo puede realizarse en la zona declarativa. Para ello se usa la palabra reservada *type* seguida del nombre del nuevo tipo, y de la palabra reservada *is*, y a continuación el conjunto de valores que pueden tomar las variables del nuevo tipo, terminado todo ello con un punto y coma.

El conjunto de valores que pueden tomar las variables se puede especificar de distintas formas. La más simple, y que da lugar a los llamados tipos enumerados, es efectivamente enumerar todos los posibles valores entre dos paréntesis. Siguiendo un ejemplo clásico, puede definirse el tipo enumerado Color con la sentencia:

type Color is (Rojo, Amarillo, Verde);

Después se pueden declarar variables y constantes de este nuevo tipo de la forma usual:

C : Color;
Semaforo : Color := Verde;
NoPasar : constant Color := Rojo;

Hay que notar que el orden con el que se especifican los valores de un tipo enumeración es significativo. De hecho, tal y como está definido el tipo Color, se puede decir que Rojo es menor que Amarillo y que Amarillo es menor que Verde. Es decir, las expresiones Rojo<Amarillo, Rojo<Verde, y Verde<Amarillo son ciertas, mientras que Rojo>Verde, etc., son falsas. En muchas aplicaciones este detalle es indiferente, pero tiene importancia a la hora de definir subtipos, tal y como se verá a continuación.

De nuevo se recuerda que debido a que Ada es un lenguaje fuertemente tipado, no está permitido, por ejemplo, asignar valores de tipo Color a variables enteras (de tipo Integer). Esto, que puede parecer evidente en este momento, sí está permitido en otros lenguajes (no merece la pena explicar cómo se realiza en este caso la conversión de tipos), lo cual permite escribir programas muy ingeniosos pero potencialmente cargados de errores. Puede verse en este caso un reflejo de las decisiones que se toman en el diseño de lenguajes de programación para alcanzar un compromiso entre flexibilidad y seguridad.

Dos tipos enumeración predefinidos muy importantes son el tipo Boolean y el tipo Character. Las variables de tipo Boolean pueden tomar sólo dos valores, true o false y se utiliza para caracterizar la veracidad de expresiones lógicas (comparaciones,...) como se verá en el capítulo dedicado a control de flujo, dentro de dos entregas. Las variables de tipo Character pueden tomar valores de entre una lista de 256 elementos. Dicha lista abarca exactamente los códigos de control y los caracteres (normales y extendidos) de la tabla ASCII. Ello incluye las letras del abecedario (mayúsculas y minúsculas), signos de puntuación, letras acentuadas, etcétera.

SUBTIPOS

Como se vio anteriormente a la hora de definir nuevos tipos es necesario establecer su conjunto de valores. La enumeración, plantea evidentes inconvenientes. Para ello es posible definir tipos a partir de otros tipos; son los subtipos que caracterizan a un conjunto de valores que es un subconjunto del conjunto de valores permitidos de otro tipo. Dicho subconjunto se define en base a una restricción o rango. Los subtipos conservan todas las operaciones de los tipos originarios. Por ejemplo, supóngase que se tiene un tipo enumeración que describa los días de la semana:

type Semana is (Lun, Mar, Mie, Jue, Vie, Sab, Dom);

Se puede especificar un subtipo que recoja los días laborables de la siguiente forma:

subtype Laborables is Semana range Lun..Vie;

Como puede verse se ha empleado la palabra reservada *range* para establecer una restricción, un rango de valores permitido dentro del tipo originario.

La declaración de variables del (sub)tipo Laborables se hace de la forma habitual. El rango que define al subtipo sólo puede abarcar valores consecutivos del abanico de valores del tipo original. Es por ello que se dijo que el orden en que aparecen los valores de un tipo enumeración puede llegar a ser importante.

Un caso especial es aquel en el que el principio del rango especificado es inferior en valor al final del rango (un ejemplo sería un rango como *Vie..Lun*, en el que *Vie>Lun*). Este caso puede producirse, e incluso no poder detectarse en tiempo de compilación, dado que en la especificación de un rango pueden aparecen variables y debido a ello sus valores podrían variar de forma potencialmente no previsible, con lo que los límites del rango se calcularían en tiempo de ejecución. En el caso de que se produzcan estos rangos, se considera entonces que el conjunto de valores del tipo es nulo.

Se pueden declarar subtipos de tipos que no sean enumerados de igual manera. Por ejemplo, un subtipo *NotaExamen* podría definirse de la siguiente forma:

subtype NotaExamen is Float range 0.0..10.0;

Y las variables:

NotaMat, NotaLeng : NotaExamen;

Ada garantiza que en ningún momento puedan violarse las restricciones establecidas en la definición del subtipo. Este concepto es muy importante. En ningún caso puede haber, por ejemplo, una variable del subtipo *NotaExamen* cuyo contenido sea un valor mayor de 10.0 o menor de 0.0.

En tiempo de ejecución se comprueba, cada vez que se modifica una variable, que su valor no se salga del rango permitido. En caso de que esto se pro-

duzca, se "levanta una excepción". Una excepción es, en general, una situación anómala que se produce durante la ejecución de un programa. El flujo de ejecución se desvía hacia código que pueda manejar la anomalía producida, código que podrá ser descrito por el programador. Las excepciones se verán detalladamente en entregas posteriores. De momento, se puede asumir correctamente que se produce un error (en este caso *Constraint_Error*, que se traduce como "error de restricción") y que el programa se detiene.

Comparando las dos cualidades que se han visto relacionadas con el tipado fuerte, se observa que la comprobación de tipos es estática, se realiza en tiempo de compilación, mientras que la comprobación de las restricciones es dinámica, se realiza en tiempo de ejecución.

Es muy importante entender que un subtipo no introduce un nuevo tipo. El tipo de las variables *Nota1* y *Nota2* es *Float*. Por ello sus valores se pueden asignar a variables *Float* y viceversa.

Incluso se puede restringir el rango de valores de una variable sin necesidad de definir un subtipo:

```
A : Integer range 0..10;
```

ATRIBUTOS

Los atributos son unos modificadores que actúan sobre un identificador (por ejemplo, sobre el nombre de un tipo o de subtipo) y que proporcionan una información que describe una característica de dicho identificador. Un atributo se escribe a continuación del identificador que modifica y precedido por un apóstrofe. Ejemplos de atributos son:

```
Integer'First
Integer'Last
```

El primero proporciona el mínimo valor que puede adoptar una variable de tipo Integer, y el segundo, el máximo. En una máquina con aritmética de 16 bits en complemento a 2, *Integer'First* proporcionará seguramente -32768 e *Integer'Last*, +32767. Los atributos *First* y *Last* pueden modificar identificadores de otros tipos o subtipos. Tienen sentido, por lo tanto, las expresiones *Float'Last, NotaExamen'First* (igual a 0.0), *Semana'First* (igual a *Lun*), etc.

Existen muchos atributos predefinidos, que se irán presentando a lo largo del curso. Algunos atributos aceptan parámetros. Tal es el caso de *Succ* y *Pred*. Ambos modifican un identificador de tipo enumerado (tipo numérico discreto o enumerado), y denotan el valor siguiente o predecesor del parámetro que se proporciona. Por ejemplo:

```
Hoy   :=Mar;
      Mannana:=Semana'Succ(Hoy);
—Mannana es Mie
Ayer  :=Semana'Pred(Hoy);  —Ayer fue Lun
Integer'Succ(2) — es igual a 3
Integer'Pred(3) — es igual a 2
```

Semana'Pred(Lun) y *Semana'Succ(Dom)* levantarían la ya conocida excepción *Constraint_Error* al intentar salirse de rango:

```
Hoy:=Dom;
      Mannana:=Semana'Succ(Hoy);
—ERROR: Constraint_Error
```

Para terminar esta breve introducción a los atributos se presentan dos más, relacionados también con tipos enumeración:

Pos(Arg): El argumento *Arg* es un valor del tipo modificado por el atributo. El valor es el de la posición de dicho argumento dentro de la lista de declaración del tipo, empezando por cero. Por ejemplo, *Semana'Pos(Lun)* es igual a 0, *Semana'Pos(Mar)* es igual a 1, etc., hasta *Semana'Pos(Dom)* que es igual a 6.

Val(Arg): Es complementario a *Pos*. El argumento es un entero y el valor es el del correspondiente al tipo cuya posición en la declaración del tipo es *Arg*. Por ejemplo *Semana'Val(Mie)* es igual a 2.

INTERACCIÓN CON EL USUARIO

En este punto del curso es necesaria alguna forma de poder interactuar con el usuario, elemento común de un gran conjunto de aplicaciones. Una posibilidad sería una librería (en forma de uno o varios paquetes Ada) para un sistema de ventanas como XWindow o MsWindows. Otra posibilidad, menos elaborada y agradable en su interfaz pero más sencilla y clara para explicar los conceptos del curso y a la vez más portable, consiste en presentar en modo texto y leer del teclado.

En caso de valores de tipo Integer o Float, se pueden usar dos paquetes proporcionados por el compilador, de la misma forma que se hizo el mes pasado para imprimir cadenas de caracteres. El listado 1 es un ejemplo de cómo utilizar éstos paquetes, que se llaman *Integer_Text_IO* y *Float_Text_IO*, para imprimir números en la pantalla. También se presenta el procedimiento *Get()* que sirve para leer del teclado un valor numérico. Asimismo aparece el procedimiento *New_Line*, del paquete *Text_IO*, que sirve para imprimir una línea en blanco. Indicar, por último, que no se han usado cláusulas use, aunque podrían haber hecho el código algo menos engorroso, para que se pueda ver a qué paquete corresponde cada procedimiento.

Para imprimir en pantalla valores de tipos enumeración, es necesario que el programador describa algún procedimiento específico que realice dicha operación. En Ada, sin embargo, existe un mecanismo que nos evita la tarea. Si se abordara el trabajo que supone escribir un procedimiento para cada tipo posible que tengamos en nuestro programa (por ejemplo, uno para imprimir variables de tipo Color, otro para variables de tipo Semana, y así sucesivamente), se observaría que todos esos procedimientos son muy parecidos y que al escribirlos se estaría repitiendo la mayor parte del código.

Este problema se plantea no sólo a la hora de escribir procedimientos para imprimir valores en la pantalla, sino en muchas tareas generales. Por ejemplo, una función que devuelve el máximo de dos valores de tipo Integer, es prácticamente idéntica a otra que devuelva el máximo de dos valores Float.

En Ada se encuentra solución a este problema permitiendo definir unas "plantillas" de código, que pueden ser luego adaptadas a distintos tipos de datos. Esas plantillas toman el nombre de paquetes o subprogramas (un subprograma es un procedimiento o una función) genéricos. La acción de especializar un paquete o subprograma

genérico a un tipo de datos determinado se conoce como instanciación. Los genéricos contribuyen a la reutilización de código, un objetivo del lenguaje que ya mencionamos el mes pasado. La forma de escribir código genérico es un aspecto algo avanzado del lenguaje y se describirá más adelante en el curso.

El paquete genérico que permite imprimir cualquier tipo enumeración se llama *Ada.Text_IO.Enumeration_IO*. En principio, extraña sintaxis de la instanciación del paquete genérico en posteriores entregas. Los procedimientos que proporciona *Enumeration_IO* son análogos a los proporcionados por *Integer_IO* y *Float_IO*.

ARRAYS

Hasta ahora hemos vista la creación de tipos nuevos por numeración y por subrango. Una tercera manera es por composición de tipos ya existentes: los *arrays* y los registros.

Los *arrays* (matrices) son objetos compuestos, formados por una serie de objetos del mismo subtipo. Un *array* puede ser unidimensional (como una lista) o multidimensional. Un ejemplo de declaración de un *array* puede ser el siguiente:

```
Lista : array (1..10) of Float;
```

Esta línea declara la variable Lista como un *array* unidimensional de 10 elementos. Cada elemento es una variable de tipo Float. Un *array* de dos dimensiones se puede declarar de la siguiente manera:

```
M3x2 : array(1..3,1..2) of Integer := ((0,0),(0,0),(0,0));
```

Se ha declarado M3x2 como un *array* bidimensional de 3 filas y 2 columnas de elementos de tipo Integer. Se muestra en este ejemplo también cómo inicializar este *array* con ceros en el momento de la declaración.

Ada permite inicializaciones más flexibles a través de la cláusula *others* o de la especificación de subrangos. Las declaraciones e inicializaciones siguientes son equivalentes:

```
subtype Rango is Integer range 1..5;
subtype Matr is array (Rango) of Integer;
```

— *Rango y Matr permiten aclarar las declaraciones:*

```
M1 : Matr := ( 0,0,0,0,0);
M2 : Matr := ( Rango'range => 0 );
M3 : Matr := ( Rango'First .. Rango'Last => 0 );
M4 : Matr := ( others => 0);
M5 : Matr := ( 0,0, others => 0);
```

Para acceder a uno de los elementos se especifica el nombre de la variable, seguido de las coordenadas, entre paréntesis, del elemento buscado:

```
M3x2(5,20):=3;
M1(Rango'First):=7;
```

Los rangos que definen las dimensiones del *array* no tienen que ser estáticos necesariamente. Pueden ser expresiones más o menos complicadas que se evalúen en tiempo de ejecución. Más interesante aún es el hecho de que los índices de un *array* no tienen por qué ser números enteros, sino que pueden ser cualquier tipo discreto. Un tipo discreto es bien un tipo entero o bien un tipo enumeración.

Por ejemplo, si queremos tener una lista de las horas de trabajo que ha fichado un empleado durante la semana, podríamos declarar el *array* horas de la siguiente manera:

```
horas : array(Semana) of Integer;
```

Entonces, para acceder un elemento, se haría:

```
horas(Lun):=8;
```

Otras ideas para declarar horas podrían ser:

```
horas2 : array(Semana range Lun..Vie) of Integer;

horas3 : array(Laborables) of Integer;
```

Es decir, el tipo y el rango de variación del índice de un *array* puede especificarse por un tipo, por un subtipo o por uno de ellos junto con un rango. De hecho,

```
Lista : array(1..10) of Integer;
```

es una abreviatura (y por lo tanto es equivalente) a:

```
Lista : array(Integer range 1..10) of Integer;
```

Existen varios atributos asociados con los *arrays*. Estos atributos modifican el nombre de una variable de tipo *array*, y nos dan información sobre sus índices:

horas'First proporciona el primer valor que puede tomar el índice, es decir, Lun.

horas'Last proporciona el último, es decir, Dom.

horas'Length denota el tamaño del rango de variación del índice, el numero de posibilidades ,en este caso 7.

horas'Range es una abreviatura de *horas'First..horas'Last* y servirá para definir un rango de variación de un índice, por ejemplo en un bucle para recorrerlo (se verán mecanismos de control de flujo en capítulos posteriores). Incluso vale para declarar variables o subtipos cuyos valores se quiere que permanezcan dentro de ese rango como ya se ha visto.

Estos cuatro atributos pueden aplicarse también a *arrays* multidimensionales. En estos casos, aceptan un argumento entero que especifica el índice al cual nos estamos refiriendo. Por ejemplo, *Matriz'Last(2)* proporcionaría el valor 2.

Los atributos también pueden modificar al identificador de un tipo *array*, siempre que sea restringido. A continuación se explicará cómo definir un tipo *array* y qué es un tipo *array* no restringido.

LISTADO 1

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
with Ada.Float_Text_IO;

procedure Ejemplo2 is
   A: Integer;
   B: Float;
begin
   Ada.Text_IO.Put("Escribe un valor entero: ");
   Ada.Integer_Text_IO.Get(A);
   Ada.Text_IO.New_Line;
   Ada.Text_IO.Put("Escribe un valor real: ");
   Ada.Float_Text_IO.Get(B);

   Ada.Text_IO.New_Line;
   Ada.Text_IO.Put("Los valores que has escrito
son: ");
   Ada.Integer_Text_IO.Put(A);
   Ada.Text_IO.New_Line;
   Ada.Float_Text_IO.Put(B);
   Ada.Text_IO.New_Line;
end Ejemplo2
```

Un tipo *array* se puede definir con una sentencia como la siguiente:

```
type Lista_10 is array (1..10) of Integer;
```

Las declaraciones de variables de este tipo se hacen de la forma habitual:

```
A,B : Lista_10;
```

Como ventaja añadida, y como antes se indicó, por el hecho de haber declarado el tipo *Lista_10* se dispone de las operaciones :=, = y != entre objetos de ese tipo. Así, por ejemplo, la asignación:

```
A:=B;
```

es equivalente (aunque más compacta) a asignar, miembro a miembro, los elementos de B a los elementos de A.

Debe indicarse que si hubiéramos hecho:

```
A,B : array(1..10) of Integer;
```

entonces la asignación A:=B no estaría permitida. Esto es debido a que según esa declaración, el tipo de A y B se considera anónimo, y en consecuencia se asume distinto para ambos.

Es posible definir un tipo *array* en el cual el rango de variación del índice no está especificado. Por ejemplo:

```
type Lista_general is array(Integer range <>);
```

El rango de variación debe proporcionarse en la declaración de variables o subtipos:

```
A : Lista_general(1..100);
subtype Lista_50 is Lista_general(1..50);

B,C : Lista_50;
```

Los tipos para los cuales el rango o los índices no están especificados, como es el caso de *Lista_general*, se llaman tipos no restringidos (*unconstrained types*). Son muy útiles en muchas circunstancias, pues permiten escribir código que va a funcionar con *arrays* de distinto tamaño.

Una peculiaridad de los tipos *arrays* es que Ada permite la conversión explícita de unos tipos a otros, siempre que los tipos de los elementos sean iguales y el de los índices también. Por supuesto, el tamaño de los rangos de variación de los índices también tienen que ser iguales. Por ejemplo, si se tiene:

```
type ABC is array(1..10) of Integer;
type XYZ is array(100..109) of Integer;
A : ABC;
B : XYZ;
```

entonces la asignación *A:=ABC(B);* está permitida.

Como ya se ha visto en un ejemplo anterior se puede asignar valores a los elementos de los *arrays* de forma más rápida y cómoda que la asignación uno por uno. Para ello se hace uso de del símbolo =>. Para explicar su funcionamiento más detalladamente trabajaremos con el *array* horas definido anteriormente de la forma:

```
horas : array(Semana) of Integer;
```

Si quisiéramos asignar a horas*(Lun..Vie)* un valor de 8 y a horas*(Sab..Dom)* un valor de 0, podemos usar el símbolo => tal y como aparece a continuación:

```
horas:=(Lun..Vie => 8, Sab..Dom => 0);
```

Es decir, se asigna al *array* horas una expresión agregada compuesta por una lista de rangos y los valores que queremos asignar a los elementos de dichos rangos. En vez de rangos podemos poner elementos individuales o listas de elementos (no necesariamente consecutivos) separados por |. Dado que se especifica el elemento al que corresponde cada valor, los elementos no tienen porqué aparecer en orden. Son equivalentes, por lo tanto, a la asignación anterior:

```
horas := (Lun..Vie => 8, Sab|Dom =>0);
horas := (Sab=>0, Lun=>8, Mar..Vie=>8, Dom=>0);
horas := (8,8,8,8,8,0,0);
```

La última línea corresponde a un agregado posicional, en el cual, al no especificarse los índices asociados a cada valor, dichos valores se asignan en el mismo orden en que aparecen a los elementos del *array*.

RECORDS

Un *record* (registro) es un objeto compuesto por una serie de elementos que pueden ser de distinto tipo. Al contrario que con los *arrays*, siempre es necesario definir un tipo *record* para poder definir una variable.

Por ejemplo, se podría definir un tipo *Estrella*:

```
type Estrella is
  record
    Distancia : Float;
    Constelacion : Nombre_Constelacion;
    Color : Color_Estrella;
  end record;
```

Donde se ha supuesto que existen definidos convenientemente los tipos agregados *Nombre_Constelacion* y *Color_Estrella*.

Los elementos agregados de una variable de un tipo *record* pueden accederse individualmente a través del nombre del campo agregado.

Así, si ahora declaramos la variable *Sol*:

```
Sol: Estrella;
```

entonces los elementos de Sol se pueden acceder como sigue:

```
Sol.Distancia :=1;
Sol.Color:= Amarillo;
```

arrays, con una expresión agregada:

```
Sol := (Distancia => 1,
        Color => Amarillo,
        Constelacion => Via_Lactea);
```

o bien, usando una expresión agregada posicional, sin necesidad de nombrar los nombres de los campos como en:

```
Sol := (1,Via_Lactea,Amarillo);
```

En el próximo número, cuya importancia es básica para el resto del curso, se explicarán los conceptos más importantes para la construcción y uso de paquetes, procedimientos y funciones.

CONCEPTOS CLAVE: tipos de datos, tipos de datos predefinidos, tipos de datos definidos por el usuario, subtipos, tipos enumeración, atributos, tipos compuestos, *arrays* y *records*, y agregados.

Contactar con los autores :
cesar@ieeesb.etsit.upm.es
bertrand@ieeesb.etsit.upm.es

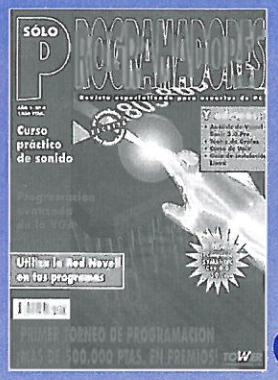
NÚMEROS ATRASADOS • NÚMEROS ATRASADOS • NÚMEROS ATRASADOS •

¡Que no te falte ni uno!

1

2

3


SÓLO PROGRAMADORES
Curso práctico de sonido
Utilizar la Red Nova en tus programas
PRIMER TORNEO DE PROGRAMACION
MAS DE 500.000 PTAS. EN PREMIOS!
4

5

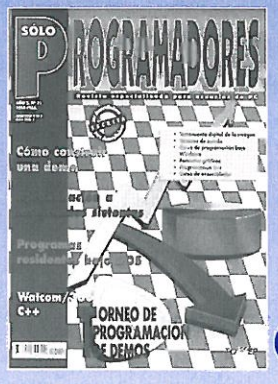
6

7

8

9

SÓLO PROGRAMADORES
DE REGALO CD-ROM con la versión shareware de Windows Sockets
Red Digital de Servicios Integrados
Terminal LINUX
Multitarea en modo protegido
10



11

SÓLO PROGRAMADORES
Estándar VESA para tarjetas SVGA
Dentro de WinEditor 2.0
DE REGALO
Demomanía: Efecto de túnel de estrellas
TORNEO DE PROGRAMACION DE DEMOS
12

13

14

15

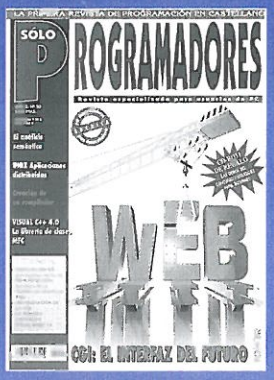
SÓLO PROGRAMADORES
Hipertexto para un sistema
Reportaje Euskal Party
El diseño de un lenguaje de programación
RAY TRACING: LA REFLEXIÓN DE LA LUZ
16

COMPLETA YA TU COLECCIÓN

17

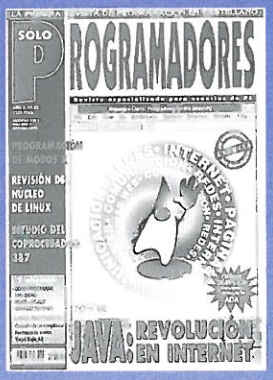
AGOTADO
18

19



20

21



22

23

24

25

26

27

28

29

Pide ahora los números que te faltan

Solicite los números que le falten enviando este cupón por correo o fax (91) 661 43 86, o llamando al teléfono (91) 661 42 11*. Horario lunes a jueves de 9 a 14 y 15 a 18h. y viernes de 9 a 15h.

SOLICITO LOS SIGUIENTES NÚMEROS ATRASADOS DE SÓLO PROGRAMADORES:

☐1 ☐2 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9 ☐10 ☐11 ☐12 ☐13 ☐14 ☐15 ☐16 ☐17 ☐19 ☐20 ☐21 ☐22 ☐23 ☐24 ☐25 ☐26 ☐27 ☐28 ☐29

Importe.....(1.250 pts. cada uno)

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta..... /.....

☐ Contra-reembolso del importe más gastos de envío.

☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.

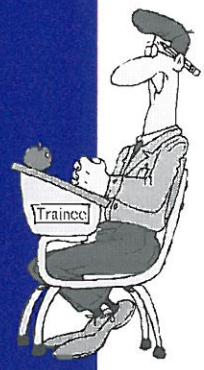
☐ Giro Postal (adjunto fotocopia del resguardo).

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Aragoneses, 7
28108 Pol. Ind. ALCOBENDAS (Madrid)

DESCRIPCIÓN DEL PUERTO SERIE

José C. Remiro



Todo PC dispone de al menos un puerto paralelo al que está destinada normalmente la conexión de la impresora y dos puertos serie, uno de ellos conectado al ratón y el otro normalmente asociado a un módem, aunque es posible conectarlo a un sinfín de dispositivos.

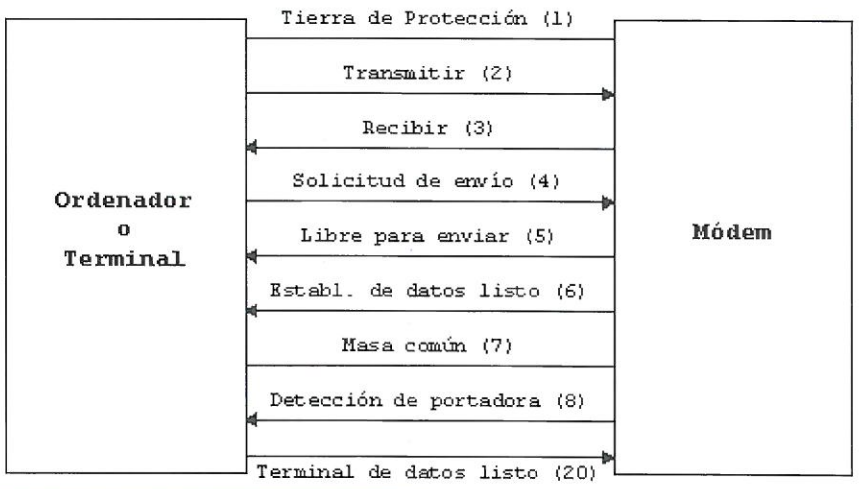
La transmisión de datos a través de un puerto paralelo, aunque mucho más rápida y fiable que la transmisión serie, presenta un claro inconveniente: no existe un medio dé transmisión tradicional que le de soporte. Por el contrario, la existencia de un medio de comunicación como lo es la Red Telefónica Conmutada, basada en la transmisión serie, junto con el reducido coste de un interfaz de tipo serie, hace que dicho puerto sea un serio candidato a utilizar como medio para poder realizar la conexión entre ordenadores.

Es evidente que la naturaleza de la representación de la información en un ordenador es de tipo discreto, mientras que la representación de la información en la RTC es de tipo analógico o continua, por lo que se debe de disponer de un dispositivo que permita la transformación de las diferentes representaciones hacia/desde el ordenador, hacia/desde la Red Telefónica; esta función la realiza el módem (Modulador-Demodulador). La interfaz que define la conexión entre un ordenador (equipo terminal de datos) y un módem (equipo terminal del circuito de datos) está determinada por la norma de la EIA RS-232-C, siendo su equivalente la norma V24/V28 del CCITT. Esta norma fija las características mecánicas, eléctricas y funcionales que debe verificar el interfaz.

Para la realización de programas que permitan trabajar con el puerto serie es imprescindible el conocimiento de las características funcionales, pues son éstas las que establecen cómo se realiza la transferencia de datos a través del interfaz, cómo se realiza el control de

Este artículo describe cómo se establece la transmisión de datos a nivel de carácter del puerto serie de un PC y cómo se puede acceder a la información y estado de dicho dispositivo.

CUADRO 1



las diversas señales en el interfaz y cómo se proporcionan las señales de sincronización que regulan la transferencia de bits. De todas estas características se hablará más adelante.

Por lo enunciado hasta ahora, es posible comprender que respaldando la comunicación entre dos ordenadores a través de sus puertos serie existen varios elementos que permiten dicha comunicación: en primer lugar el programa que se utiliza en cada uno de los ordenadores, que controla el puerto serie e interpreta los datos obtenidos a través de éste. El controlador, que es capaz de obtener la información que le llega desde el programa y disponerla como una serie de bits para su posterior retransmisión (o bien realizar el proceso inverso). El módem que transforma la señal digital, que le llega desde el ordenador, a señal analógica, utilizada por la RTC, y viceversa. Y por último, la interface que une el puerto serie con el módem.

TIPOS DE TRANSMISIÓN

Es evidente que en todo tipo de transmisión de datos debe haber una serie de convenios para coordinar las acciones del emisor y el receptor. Así, se habla de dos tipos de transmisión: asíncrona o *"Start/Stop"* y síncrona.

Cuando los n bits que forman una palabra del código utilizado van siempre precedidos de un bit denominado de arranque (o *Start*) y seguidos de al menos un bit de parada (o *Stop*) se refiere a un tipo de transmisión asíncrona; en este caso no es necesaria la existencia de una línea de transmisión para coordinar al emisor y al receptor, realizándose ésta mediante una serie de bits que van combinados entre los bytes de datos.

Cuando los datos dispensados por un terminal lo son a una velocidad fija y constante, marcada por un tiempo común a ambos y a todos los elementos que intervienen en la transmisión, se denomina transmisión síncrona; en este caso es necesaria una línea adicional a la de datos que fije el tiempo común.

Veamos cómo se produciría una transmisión de tipo asíncrono, supuesta una lógica positiva: en primer lugar la línea de transmisión podría presentar dos estados, correspondientes a 0 y 1. Si no se está transmitiendo ningún carácter, la línea se encontraría en estado 1 (*Marking contidition*); cuando modifica su estado a 0 se trata de una indicación de que comienza la transmisión de datos; a continuación se envían una serie de 5 a 8 bits dependiendo del código de caracteres utilizado; dicha información es la que realmente se desea transmitir. Es posible que a esta serie de bits le acompañe uno extra para verificar, en el extremo, la correcta recepción de la información a nivel de carácter; se trata del bit de paridad. A continuación, y puesto que ya se ha transmitido un carácter, se inserta 1, 1'5 ó 2 bits de parada (estado 1). Este proceso se repite tantas veces como caracteres halla que transmitir.

Para comprender cómo se produce la sincronización es necesario saber que antes de que se produzca cualquier transferencia de datos entre los dos extremos, éstos deben de haber establecido su velocidad de transmisión de tal forma que sea igual para los dos. Ambos extremos sabrán entonces que un bit dura el inverso de la velocidad de transmisión (1/velocidad), por lo que les basta leer de la línea cada dicho intervalo, obteniendo así cada uno de los bits que se transmiten (ahora, es posible comprender porque se puede disponer de 1'5 bits de parada).

RS-232-C

Como se comentó con anterioridad, RS-232-C se refiere a una norma que establece los aspectos mecánicos, eléctricos y funcionales del interfaz localizado entre el ordenador y el módem.

La especificación mecánica establece un conector de 25 patillas; éstas están numeradas en la fila superior de 1 a 13 (de izquierda a derecha) y de 14 a 25 en la fila inferior (de izquierda a derecha). La especificación eléctrica establece, entre otros aspectos, que para decidir un 1 binario se debe tener un voltaje más negativo que -3 voltios, y que un 0 binario se decidirá cuando se disponga de un voltaje superior a +4 voltios.

La especificación funcional indica qué circuitos están conectados a cada una de las 25 patillas, así como el significado de cada uno de ellos. Hay que decir que no todas las patillas se utilizan (de hecho, existe el conector serie simplificado DB9, con las 9 patillas más utilizadas), pues permiten realizar acciones raramente usadas; se comentarán solamente las funciones que normalmente se utilizan (ver cuadro 1).

Cuando el ordenador se enciende activa la señal *"Data Terminal Ready"* (patilla 20); cuando el módem se enciende se activa la señal *"Data Set Ready"* (patilla 6). Cuando el módem detecta una portadora sobre la línea telefónica se activa la señal *"Carrier Detect"* (patilla 8). La señal *"Request to Send"* (patilla 4) indica que el ordenador desea enviar datos, mientras que la señal *"Clear to Send"* (patilla 5) significa que el módem está preparado para

CUADRO 2

Nombre de registro	Siglas	Tipo de operación	Desplazamiento respecto a puerto base	Valor del bit 7 de LCR
Transmitting holding	THR	W	0	0
Receiver Data	RBR	R	0	0
Baudrate Divisor MSB	DLM	R/W	1	1
Baudrate Divisor LSB	DLL	R/W	0	1
Interrupt Enable	IER	R/W	1	0
Interrupt ID	IIR	R	2	-
Line Control	LCR	R/W	3	-
Modem Control	MCR	R/W	4	-
Line Status	LSR	R/W	5	-
Modem Status	MSR	R/W	6	-

R= Sólo Lectura W= Sólo escritura R/W= Lectura/Escritura

CUADRO 3

Ratio_Baudios	Factor de división	Valores de DLM	Valores de DLL
300	384	01h	80h
600	192	00h	C0h
1200	96	00h	60h
1800	64	00h	40h
2000	58	00h	3Ah
2400	48	00h	30h
4800	24	00h	18h
7200	16	00h	10h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

aceptar datos. Los datos son transmitidos mediante *"Transmit Circuit"* (patilla 2) y son recibidos mediante el *"Recieve Circuit"* (patilla 3).

EL PUERTO SERIE

Por ahora se han descrito las características del interfaz entre el módem y el ordenador; a continuación se describirán los elementos que intervienen en la conversión de palabras de datos en señales del puerto serie, función que realiza un chip especializado denominado UART *(Universal Asynchronus Receiver Transmitter)* en sus versiones 8250 y 16550, siendo ésta última una versión mejorada de la anterior.

El UART dispone de una serie de registros accesibles mediante software; la programación del puerto serie se realizará a través de éstos. Así, a modo de ejemplo, cuando el UART recibe un carácter, los diferentes bits que lo componen se almacenarán en el *Receiver Shift Register* hasta que se completa la totalidad de la palabra; tras la comprobación de la correcta recepción, la palabra pasa al *Receiver Data Register*, accesible por un programa. Cuando un determinado programa desea enviar datos, las palabras a enviar se almacenan en el *Transmitter Holding Register* y la UART las traslada al *Transmitter Shift Register* para desde aquí enviar los diferentes bits por la línea. Aunque estos registros parecen ser los más importantes, la UART dispone de otros 6 desde los que se puede inicializar y controlar su estado, cuestiones que ocuparán las siguientes secciones del artículo.

La primera cuestión a resolver consiste en el acceso a los puertos. Como es sabido, cada puerto de comunicaciones serie tiene asignada una dirección base de puerto estándar a partir de la cual se puede acceder a la información de los registros de la UART (se debe entender puerto estándar como una zona de memoria donde se realiza el intercambio de información entre un dispositivo y la CPU, mientras que el concepto de puerto serie se refiere al dispositivo como tal). Tómese como ejemplo los puertos serie COM1 y COM2: el puerto estándar para COM1 tiene la dirección 3F8h-3FFh, mientras que para COM2 la dirección es 2F8h-2FFh. En ciertas circunstancias se pueden haber cambiado dichas direcciones; sin embargo, se puede acceder a las direcciones de la BIOS 0040:0000 y 0040:0002 para conocer los puertos asignados a COM1 y COM2, respectivamente; así si el programa a desarrollar desea comprobar qué puertos están asignados a los puertos de comunicaciones deberá acceder a tales posiciones de memoria.

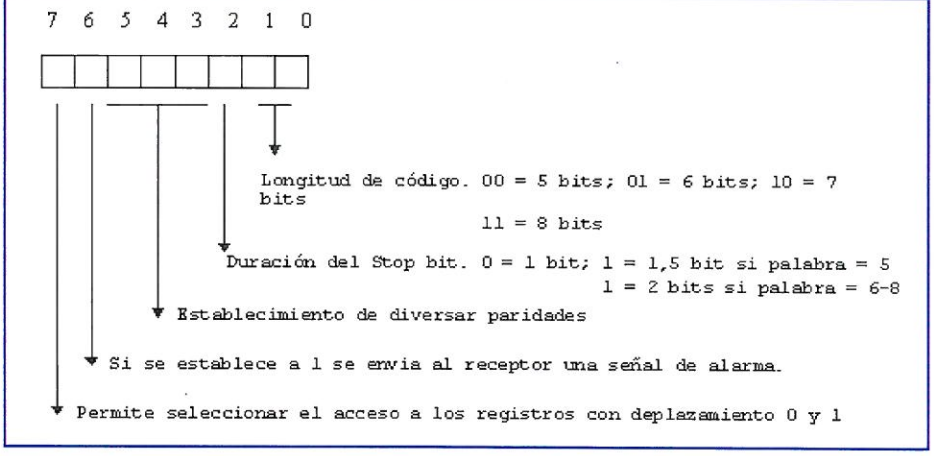
En cuanto al acceso de los diferentes registros de la UART se realiza tomando como dirección base la dirección del puerto estándar y sumando a ésta un desplazamiento, pero además para los desplazamientos 0 y 1 debe especificarse en el bit de mayor peso del *Line Control Register* un valor diferente para cada uno de ellos. En el cuadro 2, donde se describe el acceso a los registros de la UART 8250, se puede observar que al *Baudrate Divisor Register* debe accederse a partir del puerto base con un desplazamiento 0 y, además, el bit de mayor peso del *Line Control Register* debe estar a 1, mientras que para acceder a un registro que dispone de igual desplazamiento, por ejemplo el *Receiver Data Register*, dicho bit debe contener un 0. A partir de ahora, cada vez que se haga referencia a un registro de la UART se utilizará su abreviatura.

INICIALIZACIÓN Y ESTADO

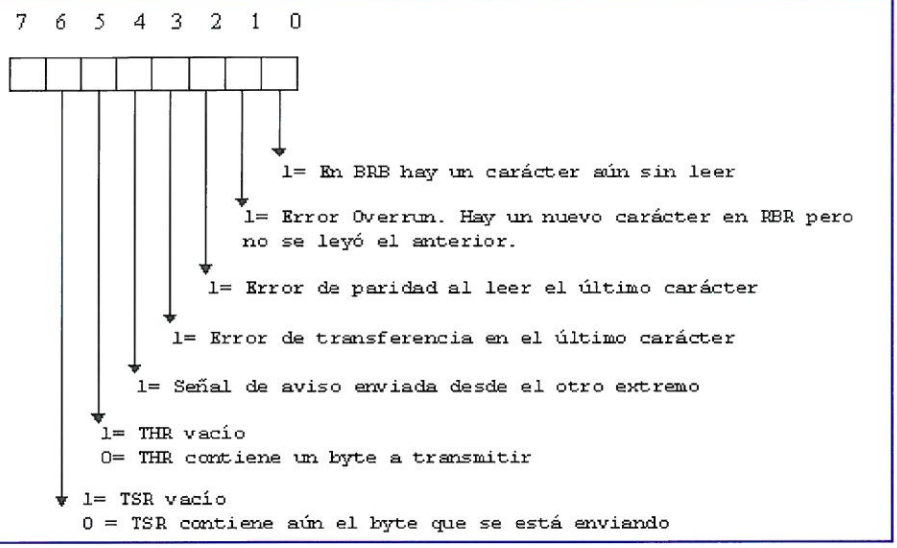
La primera acción que debe realizarse antes de iniciar una comunicación entre dos ordenadores a través del puerto serie consiste en inicializar la UART. Este proceso consiste en fijar la velocidad de la transmisión, la longitud del código de datos a utilizar, establecer el método de verificación orientado a carácter y determinar la longitud del bit de parada, conceptos todos ellos desarrollados con anterioridad.

Cada vez que se inicializa la velocidad de transmisión, la UART inicializa el resto de parámetros, por lo que en primer lugar se debe inicializar ésta. Esta velocidad no se especifica directamente, si no que se expresa en función de un cociente en relación a la frecuencia de la UART que es de 1'8432 Mhz. La fórmula a utilizar es:

CUADRO 4


7 6 5 4 3 2 1 0
Longitud de código. 00 = 5 bits; 01 = 6 bits; 10 = 7 bits
11 = 8 bits
Duración del Stop bit. 0 = 1 bit; 1 = 1,5 bit si palabra = 5
1 = 2 bits si palabra = 6-8
Establecimiento de diversar paridades
Si se establece a 1 se envia al receptor una señal de alarma.
Permite seleccionar el acceso a los registros con deplazamiento 0 y 1

CUADRO 5

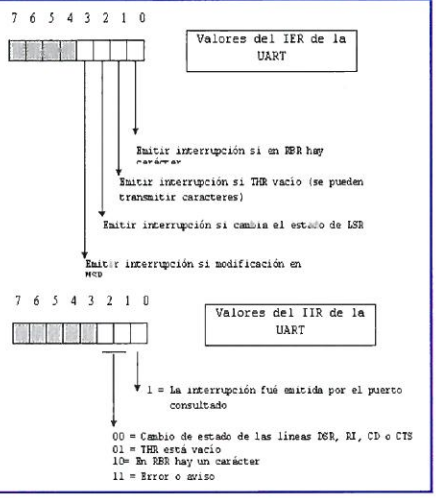

7 6 5 4 3 2 1 0
1= En BRB hay un carácter aún sin leer
1= Error Overrun. Hay un nuevo carácter en RBR pero no se leyó el anterior.
1= Error de paridad al leer el último carácter
1= Error de transferencia en el último carácter
1= Señal de aviso enviada desde el otro extremo
1= THR vacío
0= THR contiene un byte a transmitir
1= TSR vacío
0 = TSR contiene aún el byte que se está enviando

valor_del_registro = frecuencia/(16 x ratio_baudios)

que da como resultado un valor de 2 bytes; el byte de mayor peso debe situarse en DLM y el byte de menor peso debe guardarse en DLL. En el cuadro 3 se listan los valores más usuales de estos registros y su relación con la velocidad.

El resto de parámetros de inicialización enumerados anteriormente se deben disponer en el LCR (desplazamiento 3 a partir de la dirección base del puerto) y se encuentran indicados en el cuadro 4. Es interesante señalar la función del bit 6, pues a partir de él se puede generar una señal de aviso para el puerto que se encuentra en el otro extremo; por ejemplo, en el caso de que halla una situación excepcional en la que se deba interrumpir la comunicación por algún error en la transmisión. Recordar, además, que el bit 7 de este registro sirve para indicar a qué registro se desea acceder cuando dos registros presentan el mismo desplazamiento respecto a la dirección base del puerto.

CUADRO 6


7 6 5 4 3 2 1 0
Valores del IER de la UART
Emitir interrupción si en RBR hay carácter
Emitir interrupción si THR vacío (se pueden transmitir caracteres)
Emitir interrupción si cambia el estado de LSR
Emitir interrupción si modificación en MSR
7 6 5 4 3 2 1 0
Valores del IIR de la UART
1 = La interrupción fué emitida por el puerto consultado
00 = Cambio de estado de las líneas DSR, RI, CD o CTS
01 = THR está vacío
10= En RBR hay un carácter
11 = Error o aviso

En cuanto al estado de la UART, las informaciones más importantes son la indicación de emisión y recepción de caracteres y el estado de la línea de comunicación. Toda esta información puede ser conocida a través del acceso al LSR.

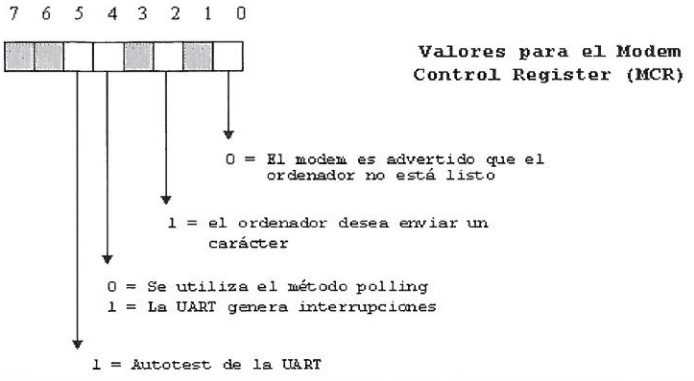
Cuando un nuevo carácter llega al RBR, el bit 0 del LSR se pone a 1; una vez que se lee dicho carácter, automáticamente, el bit vuelve a ponerse a 0. Si anteriormente a la lectura del carácter había un carácter no leído, entonces el bit 1 del LSR se pone a 1 indicando que se obvió un carácter.

Los bits 2 y 3 indican algún tipo de error en cuanto el protocolo que se estableció al principio de la comunicación como es la longitud del código utilizado, la longitud del bit de *stop* etc. Los bits 5 y 6 comunican el estado de los registros de envío de bits (TSR y THR), mientras que a través del bit 4 es posible conocer si se ha producido una señal de aviso desde el otro extremo de la comunicación.

Hay que indicar que el programa que está controlando la comunicación deberá tomar las acciones pertinentes dependiendo de la información obtenida a partir de este registro, pues por sí sola la UART no es capaz de responder a los diferentes eventos que puedan producirse a lo largo de una transmisión.

CUADRO 7

Línea	CTS	DSR	RI	RLSD	CTS	DSR	RI	RLSD
Posición de bit del MSR	0	1	2	3	4	5	6	7


7 6 5 4 3 2 1 0
Valores para el Modem Control Register (MCR)
0 = El modem es advertido que el ordenador no está listo
1 = el ordenador desea enviar un carácter
0 = Se utiliza el método polling
1 = La UART genera interrupciones
1 = Autotest de la UART

REGISTROS Y CONTROL

Es posible controlar la llegada de caracteres al puerto mediante dos estrategias: el sondeo o el control mediante interrupciones. El sondeo consiste en mantener a la CPU consultando de manera continuada la llegada y el envío de caracteres, comprobando además que no se produce otro tipo de suceso. Esta estrategia se utiliza normalmente junto con las funciones asociadas a la BIOS que se comentarán más adelante.

CUADRO 8

RATIO DE BAUDIOS

Posición de bit 7 6 5	Bits/segundo
0 1 0	300
0 1 1	600
1 0 0	1200
1 0 1	2400
1 1 0	4800
1 1 1	9600

PARIDAD

Posición de bit 4 3	Paridad
0 0	Ninguna
0 1	Impar
1 0	Ninguna
1 1	Par

BITS DE PARADA

Posición de bit 2	Número de bits de parada
0	Uno
1	Dos

TAMAÑO DEL CARACTER

Posición de bit 1 0	Tamaño del código de caracteres
0 0	-
0 1	-
1 0	7 bits
1 1	8 bits

Valores para los bits del registro AL de la función 00h

El puerto serie es capaz de emitir una interrupción cada vez que llega o se transmite un carácter, o bien cuando se produce una situación excepcional. Para informar qué evento produjo la interrupción y qué eventos se tendrán en cuenta, se utilizan los registros IIR e IER. Tras la llamada a la interrupción, entrará en acción la CPU ejecutando el administrador de interrupciones asociado, que evidentemente habrá que programar.

Por defecto, al primer puerto serie se le asigna la interrupción IRQ4 con vector de interrupción 0Ch, mientras que el segundo puerto serie tiene asignada la interrupción IRQ3 con vector de interrupción 0Bh.

A través de los cuatro primeros bits del IER se indica qué tipos de acciones del puerto serie producirán una interrupción. El cuadro 6 muestra los valores correspondientes de dicho registro.

Mediante los tres primeros bits del IIR se identifica el puerto que produjo la interrupción y la causa de la interrupción. El cuadro 6 muestra la utilización de dicho registro.

Por el momento, solamente se han descrito registros que son modificados ante la llegada o salida de caracteres al puerto y que controlan dichos sucesos; sin embargo, se puso de manifiesto que no sólo interesaban estos hechos, pues el puerto serie debía de comunicarse con el módem y había una serie de normas para la comunicación de los estados de ambos dispositivos. Existen dos registros en la UART que se encargan de este control: el MCR y el MSR (cuadro 7).

El MCR se encarga de controlar la conexión entre el puerto serie y el módem, indicando a través de sus bits el estado de las líneas RTS y DTR, mientras que el MSR registra el estado de líneas como CTS, CD, RI etc.

CUADRO 8

Bit de posición...	Registro AH	Registro AL
7	Error de exceso de tiempo	Carrier detected
6	Registro de transmisión temporal vacío	Indicador de anillo
5	Buffer de transmisión vacío	Modem preparado
4	Error de detección de ruptura	Modem preparado para transmisión
3	Error de trama	Señal de detección de portadora
2	Error de paridad	Detector de anillo cerrado
1	Error de desbordamiento	Datos preparados
0	Datos preparados	Modem preparado para enviar

Estados devueltos por el registro AX al utilizar la función 03h

EL PUERTO SERIE Y LA BIOS

Hay cuatro funciones de comunicación serie disponibles a través de la interrupción 14h (20 decimal) de la ROM-BIOS; éstos se seleccionan como es habitual en el registro AH y están numerados del 00h al 03h. El número de puerto a utilizar se introduce en el registro DX, correspondiéndose el primero de ellos con 00h.

La función 00h establece los parámetros e inicializa el puerto. En él se pueden especificar el código del ratio de baudios, el código de paridad, el código de bits de parada y el código de tamaño de caracteres; en el cuadro 8 se muestran el uso de cada uno de los bits y el valor disponible para ellos.

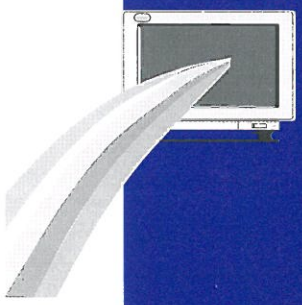
La función 01h transmite el carácter especificado en el registro AL. Si el envío fue correcto, entonces el registro AH toma el valor 00h; en otro caso, el séptimo bit tomaría el valor 1, indicando el resto de bits el tipo de error que se ha producido. De todas formas, como se verá a continuación, puede utilizarse la función 03h para obtener una descripción del error más concreta.

La función 02h recibe un carácter en el registro AL; no hay que olvidar que durante el uso de estas funciones en el registro DX se debe especificar el número de puerto que se está utilizando. Al igual que en la función 01h, en el registro AH se devuelve 00h si se recibe de forma correcta y un diagnóstico de error en caso contrario.

La función 03h devuelve el estado completo del puerto serie en el registro AX. Los 16 bits de que se compone el registro AX se divide en dos tipos de información: en AH se informa del estado de la línea (de igual forma que lo hacen las funciones 01h y 02h) y en AL se informa del estado del módem. El cuadro 9 contiene la codificación de los bits de estado.

PROGRAMACIÓN MULTITAREA EN TAPI

Carlos Arias Alonso



Tanto Windows 95 como NT permiten mediante la multitarea la ejecución de programas *multihebra*. Este término significa que dentro de una misma aplicación se pueden crear múltiples subprocesos para que se ejecuten de manera concurrente. Los mayores beneficios de esta técnica se consiguen con un sistema que soporte múltiples procesadores (por el momento sólo NT soporta más de un micro).

En el ejemplo que acompaña a este artículo, se ha decidido utilizar esta técnica, por ser la manera más eficaz de realizar la lectura de los datos recibidos a través del módem, creando una hebra o subproceso para efectuar la escucha. De esta manera, el proceso principal no tiene que estar interrogando constantemente al puerto ni se queda bloqueado esperando la recepción de datos a través del puerto, sino que permite que la rutina de procesamiento de mensajes principal de la aplicación pueda ser procesada.

En las entregas anteriores se describió el modelo de programación TAPI y se hizo un sencillo ejemplo de inicialización y manejo de las funciones básicas. En la que ahora nos ocupa, vamos a ir un poco más lejos creando una aplicación totalmente funcional que permita mantener una conversación en modo terminal entre dos ordenadores conectados a través de la línea telefónica, tener una agenda permanente de teléfonos y utilizar el ordenador como un marcador telefónico del estilo al suministrado con Windows 95. El ejecutable y el código fuente se encuentran en el CD-ROM de este mes. También se ha incluido la herramienta para desarrolladores de aplicaciones TAPI *espexe* realizada por Microsoft. Esta utilidad se proporciona totalmente gratis y se puede encontrar en Internet en el servidor de Microsoft *ftp.microsoft.com*. Dado el interés que tiene de cara a realizar las pruebas de la aplicación, se expone a continuación el modo de utilizarlo.

EMPLEO DE ESPEXE

Espexe es un emulador de dispositivos telefónicos y sirve como un excelente *debugger* para comprobar las aplicaciones TAPI. Por defecto viene con tres dispositivos de línea y dos telefónicos virtuales. Para instalarlo se abre la aplicación, se escoge la opción del menú *File* y a continuación se le da a *Install*. Conviene instalarlo antes de ejecutar cualquier aplicación que utilice TAPI, ya que si no puede funcionar inadecuadamente. Para comprobar su correcto funcionamiento abra la aplicación *marcador telefónico* que viene con Windows y seleccione la opción *Conectar usando* del menú *Herramientas*. Si entre los dispositivos disponibles figuran los tres virtuales ESP0, ESP1 y ESP2, quiere decir que se ha instalado de manera satisfactoria. En caso contrario más vale desinstalarlo y reiniciar el equipo. Una vez instalado se muestran en la ventana de la derecha todas las acciones que se van desarrollando sobre TAPI y los dispositivos virtuales.

Al comenzar la ejecución los tres dispositivos de línea están cerrados. Para poder crear sucesos y modificar sus parámetros primero hay que abrir-

En el presente artículo se acometerá la tarea de crear una aplicación totalmente funcional valiéndonos de las posibilidades de programación multitarea que ofrecen Windows 95 y Windows NT.

los mediante la función *lineOpen* desde nuestra aplicación. Cuando se encuentran abiertos se puede acceder a su información y ordenar la ejecución de sucesos sobre ellos pulsando dos veces en la ventana superior izquierda sobre la línea asociada al dispositivo. Hay que tener cuidado cuando se modifica su estado con esta utilidad, ya que, si por ejemplo se indica que pase al estado de *CONECTED* y *DIALTONE* a la vez, en ocasiones se pueden producir errores generales de la aplicación volviendo al sistema inestable.

Cuando no se desee utilizar más, hay que desinstalarlo con la opción *Uninstall* del menú. Una última puntualización; no conviene cerrar manualmente la utilidad sin antes desinstalarla, puesto que es posible que deje de funcionar si se intenta utilizar de nuevo.

DESARROLLO DE LA APLICACIÓN

Como se ha comentado anteriormente, la aplicación consiste en una práctica utilidad de conversación entre dos ordenadores en modo terminal basado en multitarea. Para construirla se ha utilizado el lenguaje C++ y las librerías MFC (*Microsoft Foundation Class*) de Microsoft, dadas las facilidades que proporcionan para crear el esqueleto de la aplicación y llevar la gestión de los distintos controles. De todos modos, en lo que respecta a TAPI, multitarea, sincronización de sucesos, solapamiento de funciones de E/S, configuración del puerto y el resto de técnicas que se explican en este artículo, se han desarrollado con el SDK de 32 bits; por tanto es fácilmente portable el código a cualquier otro compilador de C.

Los comandos de inicialización y finalización TAPI se han situado en las funciones *InitInstance* y *ExitInstance* respectivamente. Como se han explicado en artículos anteriores no hace falta comentarlas. Para la ejecución de una manera ordenada por parte del usuario de los comandos, se ha utilizado la técnica de habilitar y deshabilitar las distintas opciones de menú (ver listado 1). De esta manera, no se puede abrir la línea ni realizar ninguna otra operación sin antes haber seleccionado un dispositivo. Asimismo, mientras esté la línea abierta no se puede seleccionar un nuevo dispositivo.

LISTADO 1

```
// Habilita la apertura de línea si hay un dispositivo seleccionado
void CMainFrame::OnUpdateAbrir(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(((CTapi3App*) AfxGetApp())->m_dwDevice!=0xffffffff);
}

// Habilita el cierre del dispositivo si se encuentra abierto
void CMainFrame::OnUpdateCerrar(CCmdUI* pCmdUI)
{       // Si m_hLine != 0 hay un dispositivo abierto
        pCmdUI->Enable(((CTapi3App*) AfxGetApp())->m_hLine ? TRUE : FALSE);
}

// Habilita la realización de una llamada si:
//      Hay un dispositivo abierto (m_hLine !=0) y
// el dispositivo soporta el tipo de conexión LINEMEDIAMODE_DATAMODEM
void CMainFrame::OnUpdateLlamar(CCmdUI* pCmdUI)
{
        CTapi3App * pApp =(CTapi3App*) AfxGetApp();
        pCmdUI->Enable(pApp->m_hLine  && (pApp->m_plineDevCaps->dwMediaModes  & LINEMEDIAMODE_DATAMODEM) );
}

// Habilita la opción de colgar si se llamó anteriormente (m_hCall !=0)
void CMainFrame::OnUpdateColgar(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(((CTapi3App*) AfxGetApp())->m_hCall !=0);
}

// Habilita el envío y recepción de ficheros si:
// Se estableció la conexión y
// el dispositivo soporta el tipo de conexión LINEMEDIAMODE_DATAMODEM
void CMainFrame::OnUpdateFichero(CCmdUI* pCmdUI)
{
        CTapi3App* pApp= (CTapi3App*) AfxGetApp();
        pCmdUI->Enable(pApp->m_bConnected && pApp->
m_dwMediaMode==LINEMEDIAMODE_DATAMODEM);
}

// Habilita la selección de dispositivo si el dispositivo
// actual no está abierto
void CMainFrame::OnUpdateConfigura(CCmdUI* pCmdUI)
{
        pCmdUI->Enable(((CTapi3App*) AfxGetApp())->m_hLine ==0);
}
```

Las funciones que se muestran se encargan de activar y desactivar las opciones del menú para que el usuario realice las acciones de una manera ordenada

SELECCIÓN DE UN DISPOSITIVO

Para que el usuario pueda seleccionar el dispositivo con el que va a realizar la conexión se ha creado el recurso de diálogo *Configuración de dispositivos*. El método de inicialización de este diálogo llama al método *ShowDevices* que a su vez llama a la función global *GetDevices* para obtener información de cada dispositivo, rellenando un control *ComboBox* con los nombres de cada uno de ellos. Estos nombres se encuentran al final de la estructura de datos devuelta por *GetDevices* y se accede a ellos moldeándolos de la siguiente manera:

(LPSTR) m_pLineDevCaps+ m_pLineDevCaps->dwLineNameOffset

Finalmente llama al método *ShowPropierties* para mostrar la información detallada del dispositivo seleccionado en el *ComboBox* y a *ShowIcon* para mostrar el icono asociado.

La función *GetDevices* ejecuta el comando *lineNegotiateAPIVersion* para

LISTADO 2

```
//**********************
// LINEDEVCAPS * GetDevices (int nDevice)
//
// Devuelve información del dispositivo solicitado
// ENTRADA: hLineApp= Handle a la sesión TAPI
//                    nDevice: Dispositivo solicitado
// SALIDA: puntero a una estructura con información del dispositivo
//
// La función llamadora es RESPONSABLE DE LIBERAR LA MEMORIA apuntada por LINEDEVCAPS
//**********************
LINEDEVCAPS * GetDevices (HLINEAPP hLineApp, DWORD nDevice)
{
        DWORD dwAPIVersion;
        LINEEXTENSIONID ExtensionID;
        LINEDEVCAPS * pLineDevCaps=NULL;

        lineNegotiateAPIVersion (hLineApp, nDevice, MINVER, MAXVER, &dwAPIVersion, &ExtensionID);
        pLineDevCaps= (LINEDEVCAPS *) malloc (sizeof (LINEDEVCAPS));
        pLineDevCaps->dwTotalSize= sizeof (LINEDEVCAPS);
        // Se llama la primera vez a la función para que devuelva el tamaño de la estructura de datos
        lineGetDevCaps (hLineApp, nDevice, dwAPIVersion, 0, pLineDevCaps);
        if (pLineDevCaps->dwNeededSize > pLineDevCaps->dwTotalSize)
                realloc (pLineDevCaps, pLineDevCaps->dwNeededSize);
        pLineDevCaps->dwTotalSize= pLineDevCaps->dwNeededSize;
        lineGetDevCaps (hLineApp, nDevice, dwAPIVersion, 0, pLineDevCaps);
        if (lineGetDevCaps (hLineApp, nDevice, dwAPIVersion, 0, pLineDevCaps) !=0)
        {       // Si ocurre algún error
                free (pLineDevCaps);
                pLineDevCaps=NULL;
        }
        return pLineDevCaps;
}
```

GetDevices devuelve la información del dispositivo especificado

negociar la versión API a utilizar con cada dispositivo y llama a *lineGetDevCaps* pasando como uno de los argumentos un puntero a la estructura *GETDEVCAPS* con el campo *dwTotalSize* inicializado. Si el tamaño necesario para devolver la información es mayor (*dwTotalSize* < *dwNeededSize*) se relocaliza la estructura dinámicamente mediante la función de la librería estándar *realloc* y se vuelve a llamar a *lineGetDevCaps*. Cuando *GetDevCaps* retorna, si todo va bien, devuelve un puntero a la estructura mencionada anteriormente. Es responsabilidad de la función llamadora el liberar la señalada por este puntero.

El método *ShowPropierties* toma de la variable miembro m_*pDevCaps* la información del último dispositivo especificado con *GetDevices* y muestra la información contenida en la variable rellenando los distintos controles de la caja de diálogo.

ICONO ASOCIADO

TAPI proporciona el comando *lineGetIcon* para obtener el icono asociado a un dispositivo. Se le pasa como argumento el número de dispositivo del que se quiere obtener el icono, y devuelve un *handle* de tipo *HICON* si tiene icono, o NULL en caso contrario. En nuestra aplicación, es lo que hace el método *ShowIcon*. Para mostrar la imagen en el control *IDC_MICON* se utiliza la siguiente sentencia:

```
::SendDlgItemMessage(hwndDlg,
IDC_MICON, STM_SETICON,
(WPARAM) hIcon, 0);
```

CONFIGURACIÓN

Asociado al botón Configurar de la ventana *Configuración de dispositivos* se encuentra el método *OnConfigurar*. Esta función llama al comando TAPI *lineConfigDialog*, el cual visualiza una caja de diálogo asociada con el proveedor de servicios para poder configurar los parámetros del dispositivo. Este sistema es preferible a utilizar el par de funciones *lineGetDevConfig* y *lineSetDevConfig* para modificar los parámetros de configuración, ya que algunos de éstos son específicos del proveedor, pudiendo dejar de funcionar

LISTADO 3

```
        bLoop= TRUE;
        bRead= FALSE;       // No hay datos por leer y por tanto tiene que lanzar la lectura asíncrona
        while (bLoop)
        {

                if ( ! bRead)// Lanza la lectura asíncrona de datos del puerto serie
                        while (ReadFile (pTapiView->m_hCommPort, &dato, 1, &dwCont, &over-
lapped))
                                if (dwCont==1)
                                        pTapiView->ShowCaracter(dato);
                bRead= TRUE;
                dwWait= WaitForMultipleObjects (2, hEvents, FALSE, INFINITE);
                switch (dwWait){
                case WAIT_OBJECT_0:             // Si se ha completado la lectura del
puerto
                        if  (GetOverlappedResult(pTapiView->m_hCommPort,  &overlapped,
&dwCont, FALSE))
                                if (dwCont==1)
                                {
                                        pTapiView->ShowCaracter(dato);

                                        TRACE ("caracter=     %c\n", dato );
                                }
                        bRead= FALSE;
                        break;
                case WAIT_OBJECT_0+1:           // Finalización de tarea
                        bLoop= FALSE;
                }
        }
```

Bucle principal de la hebra creada para escuchar el puerto

si se modifican los campos de una manera inadecuada. Se recomienda utilizar las dos funciones anteriores de manera conjunta para guardar la configuración y restaurarla al estado primitivo, evitando así al usuario la molestia de realizar esta operación manualmente. Los parámetros que se pasan a la función son el número de dispositivo, el *handle* de la ventana propietaria y opcionalmente una cadena de caracteres *asciiz* con el nombre de la clase de dispositivo.

CLASES DE DISPOSITIVO. ENVÍO Y RECEPCIÓN

El modelo de programación TAPI agrupa los distintos *drivers* de dispositivo en varias clases dependiendo del método de Entrada/Salida utilizado. Este método de asociar dispositivos se conoce como agrupación por clase de dispositivo (*device class*). Cada clase tiene asociado un nombre que lo identifica inequívocamente y suministra información acerca del modelo de programación a utilizar y los comandos que pueden ser utilizados para la comunicación.

TAPI reconoce los siguientes grupos:

comm Puerto serie de comunicaciones.
comm/datamodem Módem conectado a un puerto serie.
wave/in Dispositivo de entrada de sonido Wave.
wave/out Dispositivo de salida de sonido Wave.
midi/in Dispositivo de salida midi.
midi/out Secuenciador midi (solo salida).
tapi/line Dispositivo de línea.
tapi/phone Dispositivo telefónico.
Ndis Dispositivo de red.
tapi/terminal Dispositivo Terminal.

Los proveedores de servicio TAPI pueden añadir sus propias clases si su dispositivo no se puede clasificar entre los anteriores.

En la aplicación de ejemplo se utiliza un dispositivo de la clase comm/datamodem para establecer la comunicación. Para asociar una determinada clase con el dispositivo que va a realizar la comunicación se utiliza la función síncrona *lineGetID* o *phoneGetID*. A esta función se le pasa en una cadena de caracteres *asciiz* el nombre de la clase, y en un puntero la dirección a una estructura de datos *VARSTRING*. A la salida retorna al final de la estructura anterior un valor que hay que moldear dependiendo de la clase que se pasó como parámetro de entrada. Para un dispositivo de línea al que se le asigna para la comunicación una clase "comm/datamodem" las sentencias son las siguientes:

```
VARSTRING * pDeviceID;
lineGetID(0, 0, hCall,
LINECALLSELECT_CALL,  pDeviceID,
"comm/  datamodem");
  HANDLE hCommPort= (HANDLE)
((char  *)  pDeviceID  +  sizeof
(VARSTRING));
```

En este caso se ha moldeado el dato a tipo HANDLE para la utilización con las funciones estándar de Entrada/Salida *ReadFile* y *WriteFile*

E/S ASÍNCRONA Y GESTIÓN DEL PUERTO SERIE

Tanto en Windows 95 como en NT la gestión del puerto serie se realiza mediante las funciones estándar de tratamiento de ficheros. Hay que olvidarse de las funciones del SDK de 16 bits *OpenComm, CloseComm, ReadComm* y *WriteComm*. En su lugar se utilizan las funciones *CreateFile, CloseHandle, ReadFile* y *WriteFile*. Con TAPI, la función *CreateFile* no se utiliza para la apertura del puerto, usándose en su lugar *lineGetID*. Esta función, como se vio anteriormente, devuelve un *handle* que será utilizado para el resto de operaciones con el puerto.

Los parámetros de configuración del puerto como la velocidad de transmisión, número de bits de parada, paridad, etc, se definen en una estructura *DBC* que es pasada a la función *SetCommState*. Si se quiere obtener su configuración se llama a *GetCommState*. El SDK también ofrece una función de ayuda a la configuración, *BuildCommDCB*, a la que se le pasa como parámetro una cadena de caracteres con los parámetros de configuración al estilo del comando *mode* del DOS.

Con las funciones *GetCommTimeouts*, *SetCommTimeouts* y *BuildCommTimeouts*, se configura el comportamiento del puerto serie con respecto a la recepción y escritura de datos sobre los buffers de E/S. La estructura *COMMTIMEOUTS* se pasa en las dos primeras funciones para recibir o seleccionar la configuración.

En el caso de la aplicación de ejemplo, la configuración del puerto se realiza en la función *CrearHebra*, después de llamar a *lineGetID* para obtener el *handle* del puerto. Cuando no se va a utilizar más el puerto hay que llamar a la función *CloseHandle*. Este *handle* es almacenado en la variable miembro

LISTADO 4

```
Método CrearHebra de la clase CTapi3View (tapi3view.cpp)
        /* Crea el suceso de señalización para finalizar la tarea con Reset manual y estado no señalado*/
        m_hEvEndThread= CreateEvent (NULL, TRUE, FALSE, NULL);

Función CommThreadProc de control de la hebra (tapi3view.cpp)
        HANDLE hEvents[2];
        OVERLAPPED overlapped;
        overlapped.Offset= 0;
        overlapped.OffsetHigh=0;
        overlapped.hEvent= CreateEvent(NULL, TRUE, FALSE, NULL); /// Señalizador creado para la lectura del puerto
        hEvents[0]= overlapped.hEvent;
        hEvents[1]= pTapiView->m_hEvEndThread;    //Suceso de señalización de fin de hebra
        ...
        switch (dwWait= WaitForMultipleObjects (2, hEvents, FALSE, INFINITE))
          case WAIT_OBJECT_0 :
          case WAIT_OBJECT_0 + 1 :
        ...
        CloseHandle (overlapped.hEvent);

Método KillCommRead de la clase CTapi3View (tapi3view.cpp)
        CloseHandle (m_hEvEndThread);
```

Secuencia de pasos para gestionar los objetos señalizadores

m_hCommPort de la clase vista CTapi3View.

Para la lectura del puerto se ha utilizado la función *ReadFile* en su modalidad asíncrona. Para ello se pasa como último parámetro de la función la dirección de una estructura de tipo *OVERLAPPED*. El término *overlapped* (solapado) significa que el tiempo empleado en realizar la operación con el archivo se solapa con el tiempo utilizado por la aplicación para realizar otras tareas. En otras palabras, la lectura asíncrona se realiza mediante multitarea. La hebra encargada de realizar el proceso de lectura o escritura la crea el sistema operativo al llamar a las respectivas funciones de E/S. De todos los parámetros que tiene esta estructura, sólo interesa la variable *hEvent*. El resto son variables reservadas por el sistema operativo o ignoradas para operaciones con el puerto serie.

SEÑALIZACIÓN DE SUCESOS

hEvent contiene el *handle* de un objeto *señalización*. Este tipo de objetos son utilizados en la programación multihebra para sincronizar los distintos subprocesos. La sentencia para crearlos tiene la siguiente sintaxis:

HANDLE CreateEvent (LPSECURITY_ATTRIBUTES Attributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName) ;

De todos los parámetros sólo interesan *bManualReset* y *bInitialState*. Si *BManualReset* es verdadero indica que es la propia aplicación la responsable de poner el objeto como no señalado con la función *ResetEvent*. Si es falso, es el propio sistema operativo quien lo pone como no señalado después de haberse detectado la activación del suceso. *BInitialState* indica el estado inicial, pudiendo ser señalado o no señalado. El *handle* devuelto por *CreateFile* se puede utilizar como descriptor de un objeto señalizador, pudiendo utilizar las funciones *WaitForSingleObject* y *WaitForMultipleObject* para detectar la finalización de una escritura o lectura asíncrona. La razón por la que no se utiliza en el ejemplo el propio descriptor del fichero como un objeto señalizador, sino un señalizador apuntado por el campo *hEvent* de la estructura *overlapped* es sencilla: si sobre el mismo dispositivo de E/S se realizan escrituras y lecturas asíncronas a la vez, no hay manera de saber cuál de las dos operaciones provoca la activación del señalizador. Sin embargo, si se pasa en el campo *hEvent* un señalizador distinto para cada tipo de operación, Windows ignora el descriptor del fichero y utiliza este otro para indicar que la operación de E/S se ha concluido.

La función *WaitForMultipleObject* se utiliza como se acaba de indicar para la detección de varios sucesos a la vez. Se pasa como parámetro el número de objetos señalizadores que se quieren detectar, un *array* con los descriptores de dichos objetos y el tiempo en milisegundos en que se desea intentar localizar la activación de un objeto. Si se pasa como valor la constante *INFINITE*, la función no retorna hasta que se haya detectado un suceso por lo menos (ver listado 3).

La función *GetOverlappedResult* comprueba el resultado de la operación de E/S asíncrona. Si la operación se ha completado de manera satisfactoria devuelve *TRUE*. En caso contrario se puede llamar a la función *GetLastError* para saber el código de error. Si dicho código es *ERROR_IO_INCOMPLETE* significa que la llamada se procesó correctamente pero que todavía no se ha completado la transferencia. En el esquema de la figura 4 se muestra la secuencia de pasos seguidos en la aplicación para gestionar los señalizadores.

Para indicar a la hebra cuándo tiene que finalizar se activa el señalizador *m_hEvEndThread*. De esta manera se evita la necesidad de crear *secciones críticas* (critical sections) debido a la modificación y lectura simultánea de variables de comprobación por parte de dos procesos.

MULTITAREA

Siempre que se habla de programación multitarea, se piensa que es una parcela reservada para los "gurús" o elite de programadores. Lo primero que se le ocurre a uno pensar es en la cantidad de objetos del *kernel* que hay que manejar: *mutexes*, semáforos, secciones críticas, etc. Sin infravalorar la complejidad que supone el desarrollo de una aplicación de envergadura que utilice multitarea, se ha visto cómo se puede aprovechar esta capacidad para la creación de aplicaciones sencillas. Lo único que queda por ver es cómo se crea y se finaliza una hebra.

Una hebra se crea llamando a la función *CreateThread*. Los parámetros que se le pasan son: un puntero a una estructura con los atributos de seguridad (si es nulo, se toman por defecto) el tamaño del *stack* (si es cero se asigna el mismo tamaño que la del proceso principal); un puntero a la función principal de la hebra; un parámetro de 32 bits de tipo *void* que se moldea al tipo que se necesite para enviar datos de inicialización a la hebra; ésto es, si al crear la hebra comienza inmediatamente la ejecución o si queda en suspenso, indicado con la constante *CREATE_SUSPEND*, hasta que el proceso se despierte con la función *ResumeThread*. Por último se pasa un puntero a una variable de 32 bits que recibirá un identificador devuelto por el sistema.

La tarea finaliza con *ExitThread*. Esta función es llamada de manera implícita al salir de la tarea con la sentencia *return*. Una vez se ha terminado la ejecución se puede llamar a la función *CloseHandle* para liberar el descriptor de la tarea. Por último, para saber si una tarea ha finalizado, se utiliza la función *GetExitCode Thread*. Si al retornar devuelve en la variable pasada como segundo parámetro el valor *STILL_ACTIVE* quiere decir que todavía sigue activa.

BIBLIOTECA

Windows NT Avanzado (Jeffrey Richter) Editorial McGrwawHill

Visual C++ 4 Unleashed (Viktor Toth) Editorial SAMS PUBLISHING

MAPI, SAPI, & TAPI Developer's Guide (Michael Amundsen) Editorial SAMS PUBLISHING

CONTACTAR CON EL AUTOR :
e-mail: carlosa@ran.es

CONTENIDO DEL CD-ROM

BORLAND TOOLS Y UTILIDADES DE PROGRAMACIÓN VISUAL

En el CD-ROM que se regala con el presente número de la revista se han incluido las versiones de evaluación de las últimas herramientas de Borland: Su conocido entorno de programación visual Delphi y el "buque insignia" de Borland en lo que a programación para Internet e intranets se refiere, IntraBuilder. También se ha incluido la ya habitual Guía-SP de Informáticos, los fuentes de la revista y una selección de utilidades shareware de ayuda a la programación visual.

INSTALACIÓN DE LAS BORLAND TOOLS

Las herramientas de Borland están situadas en el directorio \BORLAND del CD-ROM. De él cuelgan dos subdirectorios IB y DELPHI, que corresponden a IntraBuilder y Delphi, respectivamente.

La instalación se realiza de manera análoga para los dos entornos. Desde el subdirectorio DELPHI\INSTALL se realiza la instalación de Delphi, y desde el subdirectorio \IB la de IntraBuilder. Tras acceder a uno de los dos directorios (según el software que se vaya a instalar) se ejecutará el fichero SETUP.EXE.

Al comenzar se informa de la iniciación del asistente de la instalación y a continuación aparecerá una nueva ventana con toda la información relativa al software que va a ser instalado y otra ventana con todas las condiciones de la licencia. Tras aceptar estas dos pantallas comenzará el proceso de la instalación propiamente dicho.

Lo primero que hay que elegir es el tipo de instalación (completa, compacta o personalizada). Para los no expertos, lo recomendable es la instalación completa. A continuación se indicarán los directorios donde se instalarán y, por último, se indicará el grupo de programas donde se crearán los accesos directos. Después de configurar todos estos parámetros se pulsará sobre el botón *Install* y comenzará la instalación.

Tras la instalación se crearán los grupos de programas y los iconos correspondientes al software instalado. En Delphi, al acabar la instalación se podrá acceder a un sistema de registro *on-line* que permitirá registrarse a todos aquellos que tengan módem instalado.

UTILIDADES DE PROGRAMACIÓN VISUAL

El segundo apartado interesante de este CD es la selección de shareware de herramientas de ayuda a la programación visual que se incluye en el directorio \UTILS. De él cuelgan diversos subdirectorios, cada uno de los cuales contiene los ficheros de cada utilidad o los de instalación de la misma. En caso de que los ficheros estén descomprimidos, bastará con copiarlos todos al directorio correspondiente creado previamente en el disco duro. Si lo que aparece en el directorio es algún fichero SETUP.EXE o INSTALL.EXE, lo que se deberá hacer es ejecutar ese fichero para su instalación. La utilidad de cada una de estas herramientas se encuentra en el fichero DESCRIPT.ION del directorio \UTILS.

GUÍA-SP DE INFORMÁTICOS

Desde el pasado mes de Diciembre se puede encontrar en el CD-ROM que acompaña a la revista la Guía-SP de Informáticos. Esta guía se puede consultar con cualquier navegador de Internet (también se ofrece uno en el CD-ROM, en el directorio \BROWSERS) y está realizada en HTML y preparada para navegación *off-line*. La guía alberga una colección de *currículum* de informáticos, bien estudiantes, bien profesionales, bien tan sólo autodidactas, con una doble intención. Por un lado, y siguiendo la política habitual de Sólo Programadores (que pretende mantener a sus lectores al día en los temas imprescindibles para acceder al mercado laboral), para ayudar a encontrar trabajo a los jóvenes programadores, contribuyendo a la difusión de sus "demandas de empleo". Por otro lado, para proporcionar a las empresas un modo rápido y seguro en la búsqueda de personal, donde seguro hallará el perfil que se ajuste a la vacante existente en su empresa.

FUENTES

Por último, como viene siendo habitual, el CD-ROM contiene los archivos correspondientes a los artículos de la revista, que se pueden encontrar en el directorio \DISK30.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

BBS Y LINUX

En mi PC tengo instalado Linux y quiero comunicarme con otras BBS o sistemas de libre acceso en España que trabajen bajo Linux o Unix y se encuentren dentro del territorio nacional. ¿Conocen alguno?
¿Me podrían resolver el siguiente problema? Cuando me comunico con una BBS (que trabaja bajo Frodo en MS-DOS) y uso Minicom, conecto perfectamente, pero cuando quiero bajarme un fichero me da el mensaje "waiting receiver rz". ¿Qué puedo hacer?

José Piñero Aido
(Base Naval de Rota / Cádiz)

Aquí en la redacción sólo tenemos noticias de un BBS que trabaje bajo Linux. Se trata de SpiKa BBS (anteriormente Time BBS). Está en Madrid y su teléfono es el (91) 367 11 41.

El mensaje "waiting receiver rz" le indica que el programa remoto está esperando a que se inicie el protocolo Z-Modem para recibir. El "rz" significa Receive Z-Modem y su contrario, "sz", significa Send Z-Modem. Para activar la recepción Z-Modem pulse "Ctrl+A" y posteriormente pulse la "R". Aparecerá un menú con los posibles protocolos de comunicación. Seleccione Z-Modem y el problema estará solucionado.

CONEXIÓN DIRECTA

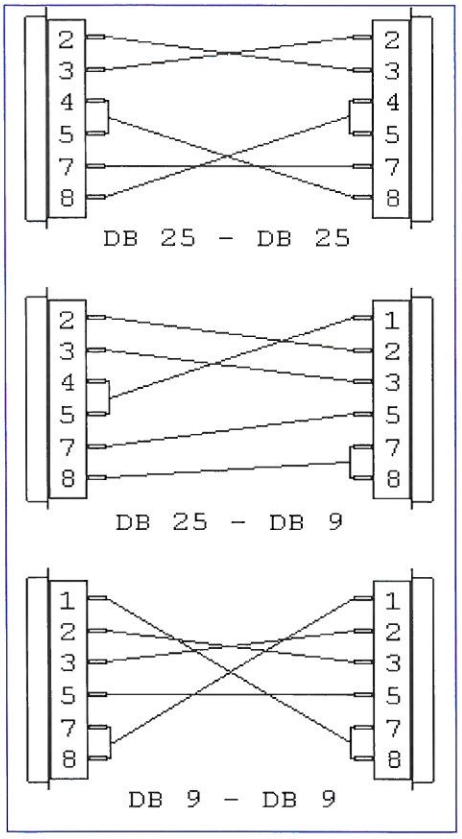
Hola, amigos de Sólo Programadores. Hace poco me he comprado un flamante Pentium y he relegado mi antiguo 486 a tareas secundarias. En un CD-ROM de una revista venía la versión shareware del increíble Quake de id Software y he pensado que, ya que tengo dos ordenadores, podría jugar en red. He visto que tiene una opción para jugar mediante "conexión directa" por el puerto serie. La documentación que acompaña al juego no explica nada al respecto. ¿Cómo puedo realizar esta conexión?

Felipe Santos Almazán
(Lérida)

Es sorprendente que los muchachos de id Software incluyan a estas alturas una opción para jugar mediante conexión directa. Al precio que están las tarjetas de red y el auge que están experimentando las redes de ordenadores, esta posibilidad se está quedando obsoleta. Pero vayamos al grano.

Para conectar dos ordenadores a través del puerto serie se necesita un cable especial denominado "cable modem nulo". El nombre viene de que el cable simula el funcionamiento de un modem frente al puerto serie del ordenador. Se trata, por supuesto, de un modem "virtual", pero lo cierto es que funciona. Actualmente es muy difícil encontrar una tienda que disponga de un cable como éste. Posiblemente no tendrá más remedio que construirlo usted mismo. Necesitará lo siguiente:

- Un cable de, al menos, 3 hilos que no supere los 15 metros de largo. Es muy importante que no supere esta longitud porque la señal se atenúa a través del cable y no llegaría en buenas condiciones al otro extremo.
- Dos conectores hembra de 9 pines. Esto suponiendo que en ambos



ordenadores tenga conectores de 9 pines en los puertos serie. Puede que necesite dos de 25 o, incluso, uno de cada. Mire la parte posterior de ambos ordenadores y compruebe qué tipo de conectores necesita.
- Un soldador de precisión, estaño y un poco de paciencia.

Todo ello lo puede encontrar en cualquier tienda de electrónica. Ahora, sólo queda indicarle cómo realizar el cableado.

Lo esencial es cruzar los pines 2 y 3 de ambos conectores De este modo, el pin de transmisión de un extremo se une al de recepción del otro extremo. También es aconsejable unir los pines de tierra (pin 5 en el caso del conector de 9 pines y pin 7 en el conector de 25 pines). Con esto sería suficiente, pero el cableado se puede mejorar mediante el uso de algunas señales de control. En la figura adjunta tiene los esquemas de las tres combinaciones posibles entre conectores.

FORTRAN 90

Soy un asiduo lector de la revista Sólo Programadores y le escribo con motivo de la aparición del lenguaje de programación Fortran 90 y su implementación en el compilador PowerStation 4.0 de Microsoft. Antes de cambiarme a C++ hace unos años yo era un programador de Fortran y solicitaría que en la revista pudiérais tratar las ampliaciones que ha habido en el paso del Fortran 77 al Fortran 90, así como una comparativa, si es posible, entre Fortran 90 y otros lenguajes (C++, Delphi, etc).

Pedro Vivas White
(Madrid)

Ante todo, muchas gracias por tu sugerencia. En Sólo Programadores siempre estamos abiertos a las propuestas de los lectores, y procuramos responder a los intereses de la mayoría. Si el tema suscita el suficiente interés, no dudes que intentaremos hacer un hueco para el Fortran 90 en la revista. Anotado queda.

PROGRAMACIÓN DE LA SB PRO

Me encanta la programación y recientemente me he interesado por el tema del sonido que me parece apasionante. Pero tengo un problema. En mi ordenador, algo anticuado en los tiempos que corren, tengo instalada una Sound Blaster Pro auténtica. Y no tengo información sobre cómo programar esta tarjeta. Por algunos artículos que he leído tengo ciertas nociones sobre programación de sonido, pero desconozco los valores del mezclador de la tarjeta. También os agradecería, si es posible, que me indicarais las direcciones de los puertos de E/S.

Julián Valenzuela Recio
(León)

No hay problema. Aquí tienes la estructura de los registros utilizados por el mezclador de la Sound Blaster Pro:

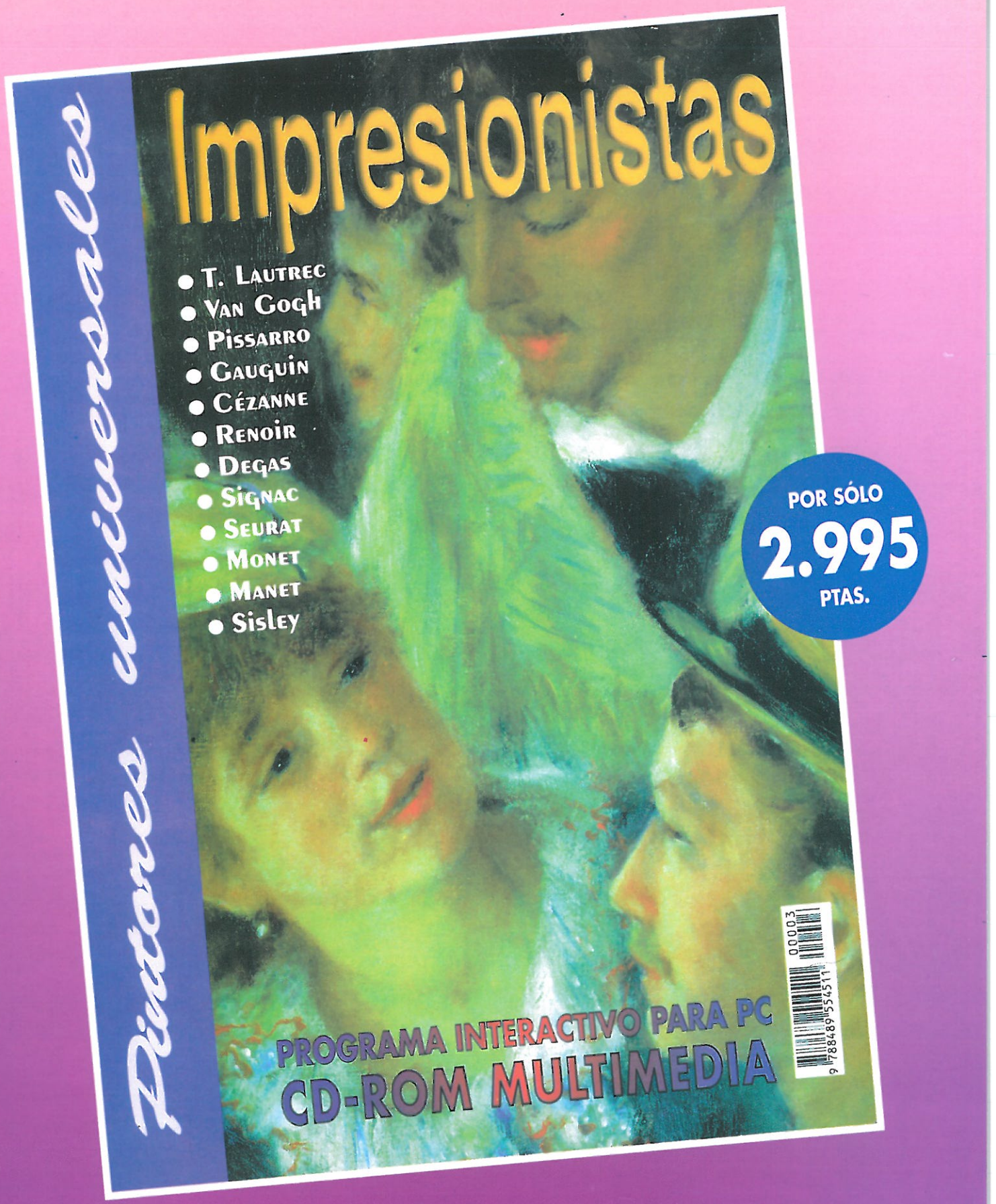
* Índice 0x00: Reinicialización del mezclador.
* Índice 0x04: Volumen de voz. Bits:
  - 7-5: Voz izquierdo.
  - 3-1: Voz derecho.
* Índice 0x0A: Volumen del micrófono (bits 2 y 1).
* Índice 0x0C: Canal de entrada. Bits:
  - 5: Filtro de entrada.
  - 3: Tipo de filtro.
  - 2-1: Fuente de entrada.
* Índice 0x0E: Varios. Bits:
  - 5: Filtro de salida.
  - 1: Estéreo activado o desactivado.
* Índice 0x22: Volumen principal. Bits:
  - 7-5: Principal izquierdo.
  - 3-1: Principal derecho.
* Índice 0x26: Volumen MIDI. Bits:
  - 7-5: MIDI izquierdo.
  - 3-1: MIDI derecho.
* Índice 0x28: Volumen del CD. Bits:
  - 7-5: CD izquierdo.
  - 3-1: CD derecho.
* Índice 0x2E: Volumen de la entrada de línea. Bits:
  - 7-5: Entrada de línea izquierdo.
  - 3-1: Entrada de línea derecho.

Recuerda que para utilizar estos registros debes utilizar dos puertos, el de dirección de registro del mezclador y el de datos del mezclador (direcciones 2x4h y 2x5h, respectivamente). La dirección concreta de ambos depende de la dirección base configurada en la tarjeta de sonido. En el primer registro se escribe el índice en el primer puerto y a continuación se escribe (o se lee) el valor correspondiente en el segundo registro.

Ya sólo te faltan las direcciones de los puertos. No olvides que la mayoría de ellos dependen de la dirección base de la tarjeta. Son las siguientes:

- 200h a 207h: Puerto de juegos. [L/E]
- Base + 0h: Puerto de estado (FM). [L]
- Base + 0h: Puerto de dirección de registro (FM). [E]
- Base + 1h: Puerto de datos (FM). [E]
- Base + 2h: Puerto de estado (FM avanzado). [L]
- Base + 2h: Puerto de dirección de registro (FM avanzado). [E]
- Base + 3h: Puerto de datos (FM avanzado). [E]
- Base + 4h: Puerto de dirección de registro (Mezclador). [E]
- Base + 5h: Puerto de datos (Mezclador). [L/E]
- Base + 6h: Reinicialización del DSP. [E]
- Base + 8h: Puerto de estado (FM). [L]
- Base + 8h: Puerto de dirección de registro (FM). [E]
- Base + 9h: Puerto de datos (FM). [L]
- Base + 0Ah: Lectura de datos (DSP). [L]
- Base + 0Ch: Escritura de comandos o datos (DSP). [E]
- Base + 0Ch: Escritura del estado del buffer (DSP). [L]
- Base + 0Eh: Estado de datos disponibles (DSP). [L]
- Base + 10h: Registro de datos (CD). [L/E]
- Base + 11h: Registro de estado (CD). [L]
- Base + 12h: Registro de reinicialización (CD). [E]
- Base + 13h: Registro de funcionamiento (CD). [E]
- 388h: Puerto de estado (FM). [L]
- 388h: Puerto de dirección de registro (FM). [E]
- 389h: Puerto de datos (FM). [E]

Entre corchetes se indica si el puerto es de lectura (L), escritura (E) o ambas (L/E).

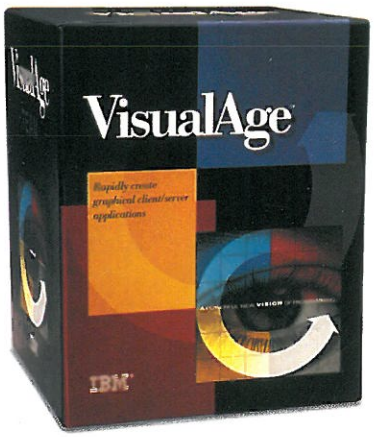


DE VENTA EN QUIOSCOS Y TIENDAS DE INFORMÁTICA

VisualAge de IBM
http://www.software.ibm.com/ad/visage

Le invitamos a conocer VisualAge.

Su empresa se lo agradecerá.



En la familia VisualAge puede encontrar: Java, C++, SmallTalk, COBOL, Basic, Generator, PACBASE y RPG.

Descubra cómo la familia VisualAge de IBM puede hacer que la información fluya fácilmente a través de su empresa y cómo opera en arquitecturas cliente/servidor multinivel. Analice las posibilidades únicas de VisualAge en desarrollo de aplicaciones con el lenguaje y sistema de su elección y en un mundo interconectado como el nuestro. Conozca casos prácticos de soluciones realizadas con VisualAge.
Visítenos en Internet: http://www.software.ibm.com/ad/visage o solicite información en el 900 100 400 de lunes a viernes de 9 a 19h.

IBM

Soluciones para nuestro pequeño mundo

Se reconocen todas las marcas. VisualAge es marca de IBM corp. (c) IBM, 1997